MSX ベーシック用語辞典

田中一郎／小山郁夫著

MSX BASIC

新星出版社

MSX ベーシック用語辞典

田中一郎
小山郁夫

新星出版社
308

MSX
シリーズ

よくわかる
ぼくらのパソコン入門

ゲームで覚える
MSXベーシック

MSX 10日間でマスター
はじめてのプログラミング

MSX 10日間でマスター
はじめてのパソコン入門

田中一郎／小山郁夫著

新星出版社

はじめに

今日のパーソナルコンピュータの普及の度合には驚かされます。

パーソナルコンピュータとは、個人で購入して使うことができるコンピュータということですが、これまでに市販されたパソコンは高価で、個人で購入するには多少の無理がありました。MSX パソコンという低価格のパソコンの出現によって、はじめてパソコンは、個人で購入して使えるパーソナルコンピュータになったといえます。

パソコンは、家庭やオフィスで、ホビーやビジネスに大活躍していますが、パソコンを実際に使ってみると、非常に優れた能力をもっていることがわかります。しかしパソコンがいかに優れた能力をもっていたとしても、しょせん機械ですから、私たちが仕事をするように命令しないかぎり仕事をしてくれるはずがありません。パソコンはすべてプログラムにしたがって働きますから、私たちがパソコンにわかる言葉でプログラムを組んで仕事の内容と手順を命令してやらなければなりません。

コンピュータのプログラミング言語としては、マシン語、アセンブリ言語、フォートラン、コボル、ベーシックなどがありますが、パソコンが直接理解して、命令を実行できる唯一の言葉はマシン語です。ほかの言葉の場合は、マシン語に翻訳されてから実行されるようになっています。ところが、マシン語は0と1でできていて、私たちにとって非常にわかりづらいもの、誰でも自由に使えるものというわけにはいきません。

そこで開発されたのが、ベーシックです。ベーシックはBASIC（Beginner's All purpose Synbolic Instruction Code）と書きます。

BASICは、人間の言葉に近いという意味から高級言語と呼ばれ、フォートランを簡単にした会話型言語で、さきにあげたプログラミング言語のなかでは、もっともやさしい、初心者向けの言葉です。

MSXタイプのパソコンをはじめとして、今日のパソコンのほとんどは、スイッチオンベーシックで、パソコンにスイッチを入れたとたん、ベーシックが使えるように作られています。パソコンのなかには、ベーシックインタプリタが組み込まれていて、ベーシックをパソコンが理解できるマシン語に翻訳して、命令を実行するようになっていますから、パソコンが普及した要因のひとつとして、初心者向けの言葉であるBASICが使える、ということも挙げられると思います。

MSXタイプのパソコンもプログラム言語としてBASICを使いますが、MSXタイプのパソコンの場合、BASICの頭にMSXをつけて、MSX－BASICと呼んでいます。このMSX－BASICの大きな特徴のひとつが、各メーカーのどの機種とも共通であるということです。

MSXタイプ以外のパソコンも、BASICでプログラムを作ることにはかわりありませんが、どの機種にも共通というものではなく、使用するBASICはあくまでその機種だけの専用語で、ある機種にあるBASICの命令が他の機種にはないとか、ある機種で作ったBASICのプログラムが他の機種では走らないとか、いろいろな問題があります。

MSXタイプのパソコンに搭載されているMSX－BASICは、各機種共通ですから、他の機種のソフトが使えないといった悩みを解決してくれ、お互いのソフトの利用も自由自在となったのです。

ところで、MSXタイプのパソコンを購入する人は、たぶん、はじめてパソコンに接する人が多いと思います。

前に、BASICはもっともやさしい、初心者向けの言葉であると書きましたが、はじめての人にとってはむずかしいものと感じられる

ことでしょうが、MSX パソコンの優れた能力を利用するためには、単なるキー操作だけでなく、MSX－BASIC を知らなければなりません。

次のプログラムをみてください。

```
10 LOCATE 18, 8        3E E9
20 PRINT "♥"           32 E4 F6
30 END                 76
```

左がベーシック、右がマシン語で書いたプログラムです。

マシン語のプログラムは、16進数で書き表わされますが、BASIC の命令はすべて英語でできています。英語といっても、学生時代に学ぶようなむずかしさはありません。なぜなら、MSX－BASIC で使う単語の数は、ほんのわずかなものだからです。このわずかな単語を、決められた書き方にしたがって並べていくと、MSX パソコンに仕事をさせる命令となるわけです。

まったく MSX－BASIC について知らなくても、うえのプログラムを見て、何を MSX パソコンにさせようとしているのか、なんとなくわかるのではないでしょうか。もちろん、コンマがなぜ使われているのか、クォーテーションマークの意味といった細かいところまでわかるはずはないでしょう。ただ、おおまかに何をさせようとしているのかがわかる、それほど BASIC はやさしいということです。

このように、MSX－BASIC で使うわずかな言葉を覚えて、決められた書き方にしたがって、この言葉を並べていくと、MSX パソコンに仕事をさせる命令が作りあげられる、ということになります。

ただし、決められた書き方を間違えると、パソコンは命令された仕事をしません。たとえコンマひとつの間違いでも、パソコンに対して命令が通じなくなりますが、コンマはこういうときに使うと決められていて、この使い方しかないというようにがっちり決められていれば、覚えるのも簡単で、また命令を書くのも簡単です。

ところで、この本は、MSX パソコンに取り組みはじめてまだ間もない人のために、また、これから MSX パソコンに取り組もうとしている人のために書きました。

したがって、MSX－BASIC という言葉のなかから、まず、この言葉を知らなければ、MSX パソコンに仕事をさせるための命令が書けないという言葉を選んで、詳しく説明しました。

また、その項で取りあげた言葉だけでなく、そこに例として示したプログラムに出てくる言葉の意味と働き、またデータがどのように処理されていくのかなど、そのつど繰り返えし説明を加えました。このように、プログラムにおいて、どうデータが処理されていくか、そのつど説明を加えたのは、こうすることによって、その項で取りあげた言葉がよく理解できると同時に、プログラムがどのように作られているかをも理解できると考えたからです。

さらに、この本は、この本の表題に示したように、辞典形式をとっています。このようにしたのは、わからない言葉がでてきたとき、そのつど、辞引きのようにわからない言葉を引いて、理解したり、使ったりすることができるようにと考えたからです。つまり、辞典形式をとったほうが、本を利用しやすいのではないかといった利用面から考えて、このような形をとりました。

この本が、MSX パソコンの優れた能力を利用するうえで、みなさんのお役に立てばと思っています。

田中一郎／小山郁夫

もくじ(索引)

ASC

ASCは、文字や記号などをキャラクタ・コードに変える

私たちが書くプログラムは、いまではそのほとんどが、ＢＡＳＩＣといってもよいようです。

ＢＡＳＩＣで書かれたプログラムは、つぎのプログラムのように、アルファベットやカタカナ、数字や記号などが、いろいろ使われています。

```
10 A=5
20 B=3
30 C=A+B
40 PRINT "ｺﾀｴﾊ";C
50 END
```


このアルファベット、カタカナ、数字、記号などを総称して、キャラクタといいます。このキャラクタのひとつひとつには、キャラクタ・コードというものがついています。

たとえば、Ａは65、Ｂは66、コは186、タは192、５は53、３は51、＝は61、＋は43といった具合です。このコードは、私たちが、ふだん使っている10進数で表わされています。

キャラクタ・コードについては、キャラクタ・コード表をみても知ることができますが、このＡＳＣを使うと、簡単にキャラクタ・コードを知ることができます。

いまここで、カタカナの〝ソ〟のコードを、知りたいとしたとしま

しょう。その場合は、つぎのようにすればよいのです。

```
10 PRINT ASC("ソ")
20 END
run
 191
```

このように、"ソ"のコードは191であることが、すぐにわかります。

ここで注意することは、ＡＳＣのカッコのなかは、必ずストリングでなければならないということです。

ですから、つぎのようにするとエラーメッセージが表示されます。

```
10 PRINT ASC(G)
20 END
run
Type mismatch in 10
```

クォーテーションマークで囲まれていないので、ストリングになっていません。("G")にします

```
10 PRINT ASC(オ)
20 END
run
Syntax error in 10
```

クォーテーションマークで囲まれていないので、ストリングになっていません。("オ")にします

もうひとつ注意することは、アルファベットの場合です。ふつうアルファベットは、小文字で入力しても、ＬＩＳＴをとると、大文字に変換されていますが、小文字をクォーテーションマークで囲むと、ＬＩＳＴしても大文字に変換されないで、小文字のままです。ですから大文字のコードを知りたいときは、ＳＨＩＦＴキーを押して、大文字で入力しなければなりません。

大文字のコードと小文字のコードは同じではなく、それぞれつぎのようになります。

```
10 PRINT ASC("A")
20 END
run
 65
```



```
10 PRINT ASC("a")
20 END
run
 97
```

小文字
小文字aのコード

前頁の10　PRINT　ASC（〝A〟）や（〝a〟)を、つぎのようにしても、そのコードを求めることができます。

```
10 A$="A"
20 PRINT ASC(A$)
30 END
```

ストリングAが＝の左側のストリング変数に入ります

行番号10のストリング変数の中身Aが、カッコのなかのストリング変数に送り込まれて、ASCでAのコードが表示されます

では、つぎのようにストリングの文字が、2つ以上並んでいるときは、どうでしょうか。

```
10 PRINT ASC("BOY")
20 END
run
 66
```

最初のBのコードだけを表示します

といったように、一番最初の文字である「シ」と「B」のコードだけしか表示しません。あとの文字は、無視されてしまいます。

では、参考までに、Bのコードを求めてみることにしましょう。

```
10 PRINT ASC("B")
20 END
run
 66
```

うえのASC（〝BOY〟）で表示されたコードと、ASC（〝B〟）で表示されたコードとは、おなじです。これで、2個以上のストリングの場合、最初のひと文字のコードしか、表示しないことがわかるでしょう。

IF～THEN～

もし天気がよければ釣りにいく、雨が降ったらとりやめ

私たちは日頃〝今度の日曜、天気だったら釣りにいこう〟とか、〝もし今度の休み、天気がよければどっかに遊びにいこう〟などといったことを、よくいいます。

いまにも雨が降ってきそうな空もようだったり、雨が降っていたのでは、釣りや遊びにでかけても、少しも楽しくないからです。釣りや遊びにでかけるには、なんといっても、天気がよいにこしたことはありません。

そこで〝天気がよかったら〟という条件をつけるわけです。そして、この条件を満たしたとき、つまり天気のよいとき、釣りや遊びにでかけます。

また運悪く条件を満たさないとき、つまり、いまにも雨が降ってきそうな空もようだったり、雨が降っていたりするときは、釣りや遊びにいくことはやめて、別のことをするといったことになります。

パソコンにも、これと似たようなことをさせることができます。そのときに使うのが、ＩＦ～ＴＨＥＮ～です。

ＩＦ～ＴＨＥＮ～は、もし（ＩＦ）～ならば（ＴＨＥＮ）～せよ、といった意味で、ＩＦ～の～に示されている条件を満たしたとき、ＴＨＥＮ～の～に示されていることを実行する、といったものです。

また、ＩＦ～の～に示されている条件を満たさないときは、ＴＨＥＮ～の～に示されていることは実行しないで、つぎの行番号に進んで、その行番号に示されていることを実行します。

このようなＩＦ～ＴＨＥＮ～の働きから、これを条件つき飛び越しとか、条件つきジャンプとか呼んでいます。なぜ、こう呼ぶかは、実際にプログラムをとりあげたところで、おはなしします。

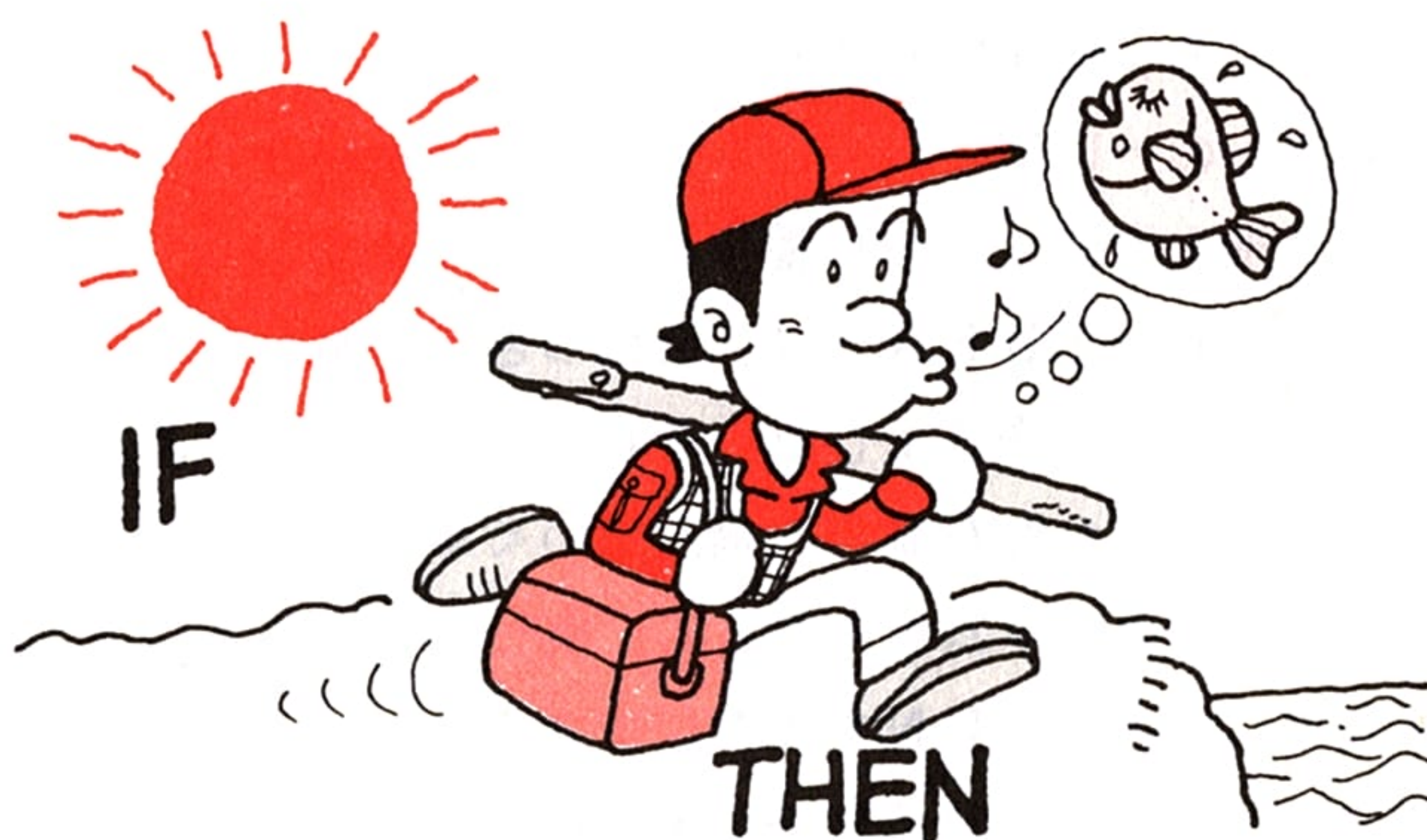
このＩＦ～ＴＨＥＮ～に、釣りや遊びのことをあてはめてみると、

ＩＦ　天気がよい　ＴＨＥＮ　釣りにいく
ＩＦ　天気がよい　ＴＨＥＮ　遊びにいく

といった具合になります。

つまり、ＩＦ～の～に示されている〝天気がよい〞という条件を満たしたとき、ＴＨＥＮ～の～に示されている〝釣りにいく〞、〝遊びにいく〞といったことが実行されるわけです。

万一、雨が降ったりしたときは、ＩＦ～の～に示されている〝天気がよい〞という条件を満たしていませんから、ＴＨＥＮ～の～に示されている〝釣りにいく〞、〝遊びにいく〞といったことは実行されずに、つぎの行番号に進む、といったことになります。

もちろんパソコンは、天気がよいなどといったような条件を、判断することはできません。パソコンが扱うものは数値ですから、数値に関する条件判断、つまり、ある数値とある数値とを比較して、ある数値がある数値よりも、大きいか小さいかを判断するといったことになります。

IF～THEN～

では実際に、簡単なプログラムをとりあげて、おはなしすることにしましょう。

```
10 PRINT "ｱﾅﾀﾉﾄｼﾊｲｸﾂﾃﾞｽｶ"
20 INPUT X
30 IF X>18 THEN 60
40 PRINT "ｱﾅﾀﾊﾏﾀﾞｺﾄﾞﾓﾃﾞｽ"
50 GOTO 70
60 PRINT "ｱﾅﾀﾊﾓｳｵﾄﾅﾃﾞｽ"
70 END
```


このプログラムでは、行番号30にＩＦ～ＴＨＥＮ～が使われています。

```
30 IF X>18 THEN 60
```

この場合の条件は、ＩＦ～の～にある〝Ｘ＞18〟です。

ここで、Ｘの中身のある数値とある数値が比較されて、条件にあっているかどうか、判断されるわけです。

〝Ｘ＞18〟は、Ｘが18より大きいこと、ということです。

したがって、Ｘに入れられた値が18と比較されて、18より大きいと判断されたとき、条件を満たしていることになって、ＴＨＥＮ～の～に示されている、60を実行するのです。

ＴＨＥＮ～の～に60といったように、数字が示されているとき、その数字は、飛んでいく先の行番号を表しています。

まえに、ＩＦ～ＴＨＥＮ～は、条件つき飛び越しとか、条件つきジャンプと呼ばれると書きました。それはこのように、ある条件を満たしたとき、指定された行番号へ飛んでいくからなのです。

では、このプログラムを入力して、実行させてみましょう。プログラムを入力したら、ＲＵＮを入力します。

するとパソコンは、行番号10を実行します。

```
10 PRINT "ｱﾅﾀﾉﾄｼﾊｲｸﾂﾃﾞｽｶ"
```

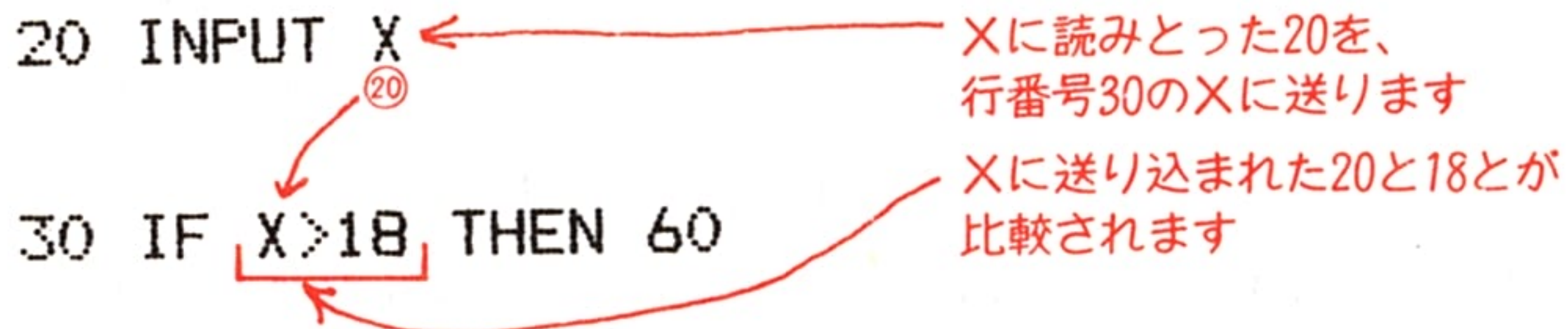
「〃」クォーテーションマークで囲まれた文字は、ストリングといって、パソコンはそのとおりにその文字をメモリのなかに記憶します。ＰＲＩＮＴは、ディスプレイに表示しなさいということですから、パソコンは、その文字をディスプレイに表示します。

行番号20のＩＮＰＵＴ　Ｘは、Ｘの値を読み込みなさいということです。そこでパソコンは、ディスプレイに？マークを表示して、Ｘに入る値を入力するよう、私たちに要求してきます。

この場合は、つぎのようにディスプレイに表示されます。

さて、行番号30ですが、そのまえに、パソコンが？マークを表示して要求しているＸの値を、入力してやらなければなりません。Ｘの値を入力してやらないとパソコンは、行番号30の実行にかかれないからです。

あなたの年令を20歳として、20をＸの値として入力しましょう。20を入力したら、必ずＲＥＴＵＲＮキーを押します。

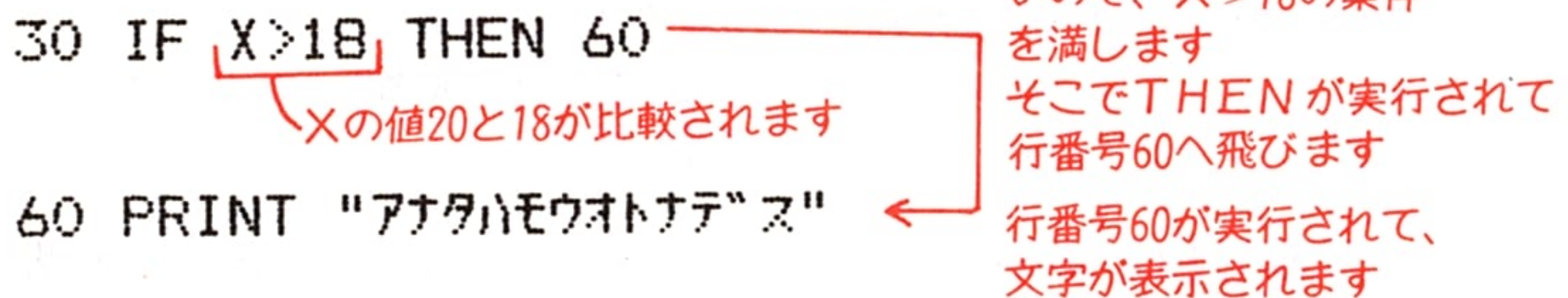
するとパソコンは、この値20をＩＮＰＵＴ　ＸのＸに読み込んで、つぎにこの値を、行番号30のＩＦ　Ｘ＞18のＸに送り込みます。

行番号30のＸに送り込まれた値20は、Ｘ＞18で、18より大きいか、どうか比較され、判断されます。20は18より大きいので、Ｘ＞18という条件を満たします。そこでＴＨＥＮ　60の60が実行されて、行番号60へ飛びます。

```
30 IF X>18 THEN 60

60 PRINT "ｱﾅﾀﾊﾓｳｵﾄﾅﾃﾞｽ"
```

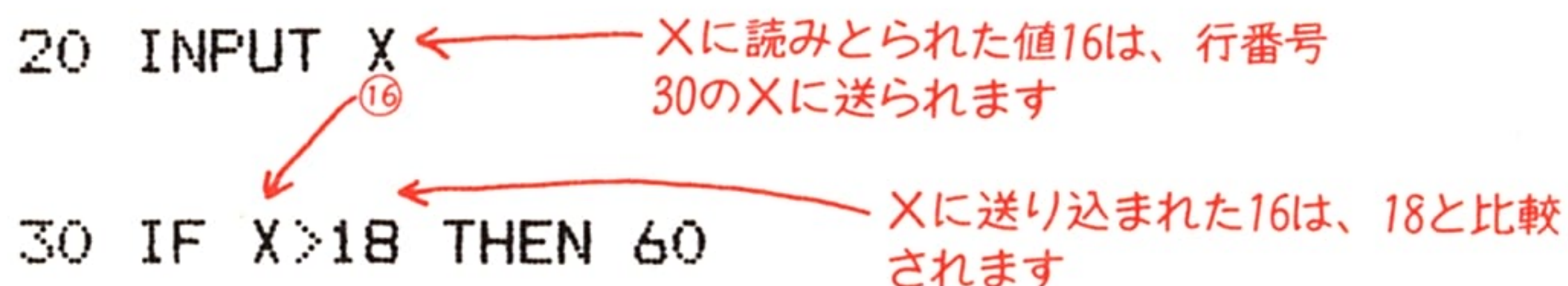
Xの値20と18が比較されます

Xの値20は18より大きいので、X＞18の条件を満します
そこでTHENが実行されて行番号60へ飛びます

行番号60が実行されて、文字が表示されます

行番号60は、行番号10と同じです。行番号60が実行されて、「アナタハモウオトナデス」が、ディスプレイに表示されます。

```
ｱﾅﾀﾊﾓｳｵﾄﾅﾃﾞｽ
```

つぎに行番号70　ＥＮＤ（おわり）に進み、パソコンは、これでプログラムの終了を知って、ＯＫをディスプレイに表示します。

では、Ｘ＞18という条件を満たさないときは、どうなるでしょうか、それをみてみましょう。今度は、18より小さい数16をとって、入力します。入力の方法は、まえと同じなので省略します。

ＩＮＰＵＴ　Ｘに読みとられた値16は、行番号30のＸ＞18のＸに送

られ、18より大きいかどうか比較判断されます。16は18より小さいので、X＞18の条件を満していません。そのため、ＴＨＥＮ　60の60は実行しないで、つぎの行番号40へ進みます。

行番号40は、行番号10や60と同じですから、これが実行されて、「アナタハマダコドモデス」が、ディスプレイに表示されます。

そして、行番号50へ進みます。行番号50は、ＧＯＴＯ70です。

ＧＯＴＯとは、無条件ジャンプといって、ＧＯＴＯのあとに示されている行番号へ飛んでいきます。そして、飛んでいった先の行番号にあるステートメントを実行します。

この場合は、ＧＯＴＯ70ですから、行番号70へ飛んでいきます。

行番号50でＧＯＴＯを用いたのは、ここになんの方法も講じないで、つぎのようにプログラムを作ったとすると、

```
40 PRINT "アナタハマダコドモデス"
50 PRINT "アナタハモウオトナデス"
```

パソコンは、行番号40を実行して、アナタハマダコドモデスを表示すると、つぎに行番号50に進んでしまい、アナタハモウオトナデスをも、一緒に表示してしまうからです。

では、つぎにＩＦ～の～に入る比較判断の記号について、まとめておきましょう。

記号	使い方
＝ 等しい	ＩＦ Ａ＝Ｂ ＴＨＥＮ 1000 もし ＡがＢと等しい ならば 行番号1000へ飛べ
＜ 小さい	ＩＦ Ａ＜Ｂ ＴＨＥＮ 1000 もし ＡがＢより小さい ならば 行番号1000へ飛べ
＞ 大きい	ＩＦ Ａ＞Ｂ ＴＨＥＮ 1000 もし ＡがＢより大きい ならば 行番号1000へ飛べ
＜＝ 等しいか小さい	ＩＦ Ａ＜＝Ｂ ＴＨＥＮ 1000 もし ＡがＢと等しいか ＡがＢより小さい ならば 行番号1000へ飛べ
＞＝ 等しいか大きい	ＩＦ Ａ＞＝Ｂ ＴＨＥＮ 1000 もし ＡがＢと等しいか ＡがＢより大きい ならば 行番号1000へ飛べ
＞＜ 等しくない	ＩＦ Ａ＜＞Ｂ ＴＨＥＮ 1000 もし ＡとＢが等しくない ならば 行番号1000へ飛べ

なお、つぎのようにＩＦ～の～に計算式が入るときもあります。

ＩＦ Ａ＝Ｂ＊Ｃ ＴＨＥＮ 1000
もし ＡがＢＸＣの値と等しい ならば 行番号1000へ飛べ

ＩＦ Ａ＞Ｂ／Ｃ ＴＨＥＮ 1000
もし ＡがＢ÷Ｃの値よりも大きい ならば 行番号1000へ飛べ

ＩＦ～ＴＨＥＮ～のかたち

ＩＦ 条件 ＴＨＥＮ 行番号または命令文

IF～THEN～ELSE

天気だったら、釣りにいく、雨だったら、映画にいく

ＩＦ～ＴＨＥＮ～の項でおはなししたＩＦ～ＴＨＥＮ～は、天気がよいときのことだけを決めたものでした。

ＩＦ　天気がよい　ＴＨＥＮ　釣りにいく

ＩＦ　天気がよい　ＴＨＥＮ　遊びにいく

もし、その休みの日に、不運にも雨が降っていたときは、いきあたりばったりで、そのときのなりゆきにまかせる、といったものでした。

しかし、このＩＦ～ＴＨＥＮ～に、ＥＬＳＥといったものがつくと、事情はまったく変ってきます。

ＥＬＳＥとは、さもなくばといった意味です。したがって、ＩＦ～ＴＨＥＮ～にＥＬＳＥ～がつくと、

ＩＦ　天気がよい　ＴＨＥＮ　釣りにいく　ＥＬＳＥ　映画にいく
もし　ならば　さもなくば

といったことになって、天気がよいときのことだけではなく、雨が降ったときのことも、決めておくことになります。

つまり、ＩＦ～の～に示されている天気がよいという条件を満たしたとき、ＴＨＥＮ～の～に示されている、釣りにいきます。また、ＩＦ～の～に示されている天気がよいという条件を満たしていないときは、ＥＬＳＥ～の～に示されている、映画にいくといったことになるわけです。

これを、パソコンにやらせるためのプログラムにすると、

```
10 INPUT A,B
20 IF A>B THEN PRINT "ツリ" ELSE 40
30 GOTO 10
40 PRINT "エイガ"
50 GOTO 10
60 END
```

？マークを表示して、ＡとＢの値を求めます。このＡとＢの値は行番号20のＡとＢに送られます

ＡとＢの値が比較され、ＡがＢより大きいかどうか判断されます
ＡがＢより大きいときＴＨＥＮのあとのＰＲＩＮＴ"ツリ"が実行されます
ＡがＢより小さいときは、ＥＬＳＥのあとの40が実行されて行番号40に飛びます

行番号10にジャンプ

エイガを表示します

行番号10にジャンプ

といった具合になります。

パソコンは、数値を扱うものですから、私たちが天気を判断するといったような複雑なことはできません。そこで、数値の大小の比較をを用いました。また釣りにも、映画にもいくことができませんから、釣り、映画を表示させることにしたわけです。

行番号20の

```
IF A>B THEN PRINT "ツリ" ELSE 40
```

の条件は、ＩＦ　Ａ＞Ｂですから、Ａの値がＢの値より大きいこと、です。したがって、Ａの値がＢの値より大きいとき、条件を満たしたことになって、ＴＨＥＮ～の～に示されているＰＲＩＮＴ〝ツリ〟を実行します。

また、この逆にＡの値がＢの値より小さいときは、Ａ＞Ｂの条件を満たしていませんから、ＴＨＥＮ～の～に示されていることは、実行しないで、ＥＬＳＥ～の～に示されていることを、実行します。つまり、この場合は、ＥＬＳＥ　40ですから、行番号40へ飛んでいくわけです。

ＩＦ～ＴＨＥＮ～の項では、ＩＦ　Ｘ＞18　ＴＨＥＮ　60として、ＴＨＥＮ～の～に60といった、行番号が用いられている例でおはなししましたが、うえの例のように、ＴＨＥＮ　ＰＲＩＮＴ～といった具合に、命令文を用いることもできます。

もちろんＥＬＳＥ～の～にも、行番号ばかりでなくＥＬＳＥ　ＰＲ

ＩＮＴ～といったように、命令文を用いることができます。

では、プログラムを実行させてみましょう。ＲＵＮを入力するとパソコンは、行番号10を実行して、ＩＮＰＵＴのあとにつづく変数ＡとＢの値を求めてきます。

ＩＮＰＵＴの項で、ＩＮＰＵＴは？マークを表示して、変数ＡとＢの値を求めてくることは、おはなししました。

ここでは、まずＥＬＳＥの働きをみるために、条件を満たさない値を入力します。

条件はＡ＞Ｂでしたから Ａが Ｂより小さければ、条件を満たさないわけです。そこでＡを１、Ｂを５として入力します。

１と５を入力して、ＲＥＴＵＲＮキーを押すと、１はＡへ、５はＢに読みとられます。ＡとＢに読みとられた１と５は、行番号20のＡとＢに送られます。

IF～THEN～ELSE～

```
10 INPUT A,B
```
①⑤

```
20 IF A>B THEN PRINT "ツリ" ELSE 40
```

ＩＦ　Ａ＞ＢのＡ＞Ｂで、値の大小が比較され、条件が判断されます。この場合は、Ａの値１とＢの値５です。Ａ＞Ｂは、Ａの値がＢの値より大きいこと、です。Ａの値１とＢの値５では、ＡがＢより小さいので、条件は満たしていません。したがって、ＴＨＥＮ　ＰＲＩＮＴ〝ツリ〟は実行しないで、ＥＬＳＥ　40を実行します。

ＥＬＳＥのあとに行番号がくると、その行番号へ飛べということになりますから、行番号40へ飛んで、行番号40を実行します。

```
20 IF A>B THEN PRINT "ツリ" ELSE 40
```
Aの１はBの５より小さいのでＥＬＳＥ40が実行されます。そして行番号40へ飛びます

```
40 PRINT "エイガ"
```

行番号40は、ＰＲＩＮＴ　〝エイガ〟です。〝エイガ〟といったように、文字が〝〃〟クォーテーションマークで囲まれているものは、ストリングといって、そのとおりに文字がメモリのなかに記憶されます。ＰＲＩＮＴは、表示しなさいということですから、エイガという文字が、ディスプレイに表示されます。

```
run
? 1,5
エイガ
```
入力したAとBの値

行番号40を実行するとストリングを表示します

パソコンは、エイガをディスプレイに表示すると、行番号50に進みます。行番号50は、ＧＯＴＯ　10です。

ＧＯＴＯは、無条件ジャンプといって、ＧＯＴＯのあとに指定されている行番号に、無条件に飛んでいきます。そして、また、行番号10のＩＮＰＵＴ　Ａ、Ｂを実行して、ＡとＢの値を入力するよう、？マークを表示してきます。

ではつぎに、Ａ＞Ｂの条件を満たしたときの働きをみてみましょう。Ａ＞Ｂの条件を満たすのは、Ａの値がＢの値より大きければよいわけですから、Ａの値を６、Ｂの値を３とします。入力の仕方は、まえとおなじなので、省略します。

入力したＡの値６と、Ｂの値３は、行番号10　ＩＮＰＵＴ　Ａ、ＢのＩＮＰＵＴによって、ＡとＢに読み込まれます。読み込まれたＡの値６とＢの値３は、つぎに行番号20のＡとＢに送られます。

```
10 INPUT A,B
         ⑥ ③

20 IF A>B THEN PRINT "ツリ" ELSE 40
```

前回と同じように、ＩＦ　Ａ＞ＢのＡ＞Ｂで、Ａの値６とＢの値３が比較され、Ａ＞Ｂの条件を満たしているかどうか判断されます。

Ａの値６は、Ｂの値より大きいので、ＡがＢより大きいこと、という条件を満たしていますから、ＴＨＥＮ　ＰＲＩＮＴ〝ツリ〟を実行

します。

```
20 IF A>B THEN PRINT "ツリ" ELSE 40
```

Aの値6とBの値3が比較されます。この場合A＞Bの条件を満たすのでTHENを実行します

ＴＨＥＮ　ＰＲＩＮＴ〝ツリ〟が実行されると、ディスプレイには、つぎのように表示されます。

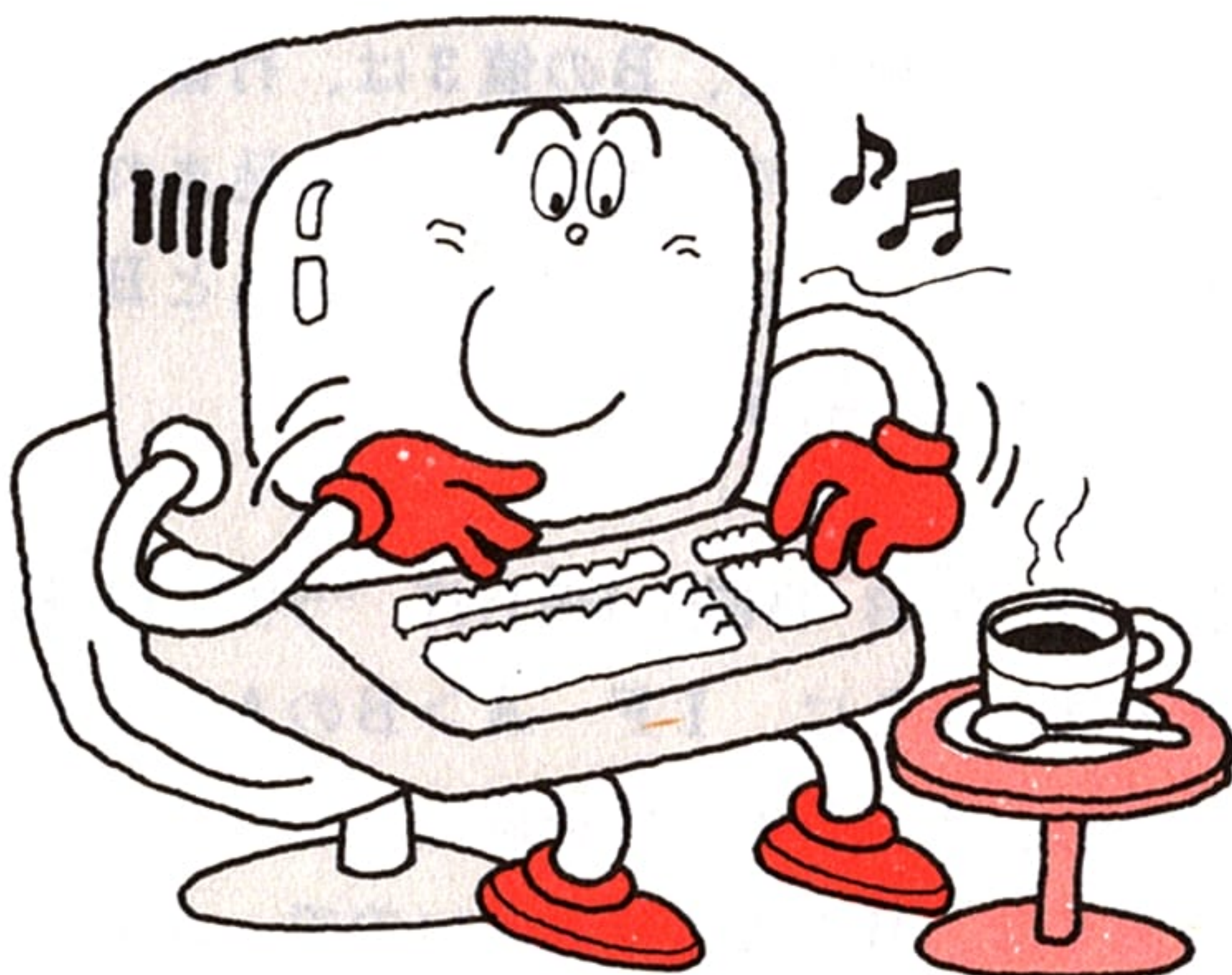
```
run
? 6,3
ツリ
```

入力したAとBの値

THEN　PRINT〝ツリ〟の実行結果です

このようにＩＦ～ＴＨＥＮ～ＥＬＳＥ～は、ＩＦ～の～で示された条件を満たしたとき、ＴＨＥＮ～の～で示されたことを実行し、条件を満たしていないとき、ＥＬＳＥ～の～で示されたことを、実行するというものです。

ＩＦ～ＴＨＥＮ～ＥＬＳＥのかたち

ＩＦ　条件　ＴＨＥＮ　行番号または命令文
ＥＬＳＥ　行番号または命令文

CLS

CLSは、画面に表示されているものをすべて消す

CLSは、画面に表示されているものを、すべて消す働きをします。つぎのプログラムは、球がピンポン球のように跳返えって飛ぶというものです。このプログラムでは、CLSが行番号10と、行番号50に使われています。

```
10 CLS
20 LOCATE 10,10
30 PRINT "タマ ノ イト゛ウ"
40 FOR T=1 TO 2000:NEXT T
50 CLS
60 X=0:Y=10
70 XX=1:YY=1
80 X=X+XX:Y=Y+YY
90 LOCATE X,Y:PRINT "●"
100 IF X=1 THEN XX=1
110 IF X=27 THEN XX=-1
120 IF Y=1 THEN YY=1
130 IF Y=21 THEN YY=-1
140 LOCATE X,Y:PRINT " "
150 GOTO 80
```

このプログラムを実行させると、行番号10の CLS が実行されて画面に表示しているものをすべて消します。

行番号10で CLS を使っているのは、行番号20のLOCATE で指定して、行番号30のPRINT で「タマノ　イドウ」というタイトルを画面の真中に表示するためです。タイトルを表示するとき、画面に何も表示されていないほうがすっきりと表示することができるので、画面に表示されているものをすべて消しているわけです。

行番号40のFOR～NEXTはウエイト・ループで、タイトルを表示しておく時間を決めています。T＝1　TO　2000ですから、変数Tが初期値の1から最終値の2000になるまで、FORとNEXTの間をいったりきたりします。2000をこえると、行番号50の CLS を実行して、タイトルを消します。

行番号50の CLS で、画面に表示されているタイトルを消すのは、球が跳返えって飛ぶとき、タイトルが邪魔になるからです。そこで、CLSでタイトルを消すわけです。

行番号60から150までのステートメントが、球を飛ばして、球が画面の端にくると跳返えすステートメントです。ステートメントの働きについての説明は、ここでは省略します。

このようにCLSは、画面に表示するものをすっきり表示させるために、不要になった表示を消すために使います。

ところで、さきのプログラムを実行させると、画面の下のほうに、colorとかautoとかいった、ファンクションキーの内容が表示されたままになっています。この表示もじゃまになるときがあります。このファンクションキーの表示を消すには、CLSだけでは消すことができません。

CLS：KEY　OFF

というように、CLSのあとにコロン「：」で区切って、KEY　OFFをつけると、ファンクションキーの表示も消すことができます。

INT(X)

INTは、整数を作る

パソコンには、いろいろな関数が組み込まれています。関数というと、数学ととらえてしまって、敬遠する人もいるかも知れませんが、パソコンに組み込まれている関数は、数学としてとらえてとやかくいっても、はじまらないものなのです。

なぜなら、それぞれの関数につづく（X）のなかのXに、値を入れれば、求める値が得られるようになっているからです。ですから要は、パソコンに組み込まれている、いろいろな関数の働きを知って、その活用の仕方をおぼえておけば、それで十分なのです。

ＩＮＴ（Ｘ）も、パソコンに組み込まれている関数のひとつです。いい忘れましたが、パソコンに組み込まれている関数を、組み込み関数と呼んでいます。

ＩＮＴは、 integer の略で、整数という意味です。そして、その意味のとおり、ＩＮＴ（Ｘ）のＸに、小数点をともなった値を入れると、ＩＮＴという、関数処理を行って、整数を返します。つぎのようにです。

```
10 INPUT X
20 S=INT(X)
30 PRINT S
40 GOTO 10
50 END
```

ＩＮＰＵＴ　Ｘは、Ｘの値を読みとりなさい

行番号10で読みとられたＸの値は、ＩＮＴ（Ｘ）のＸに送り込まれます
このＸの値は、ＩＮＴによって関数処理が行われて、その結果が＝の左のＳに入ります

このＳに入った値は、行番号30のＳに送られます

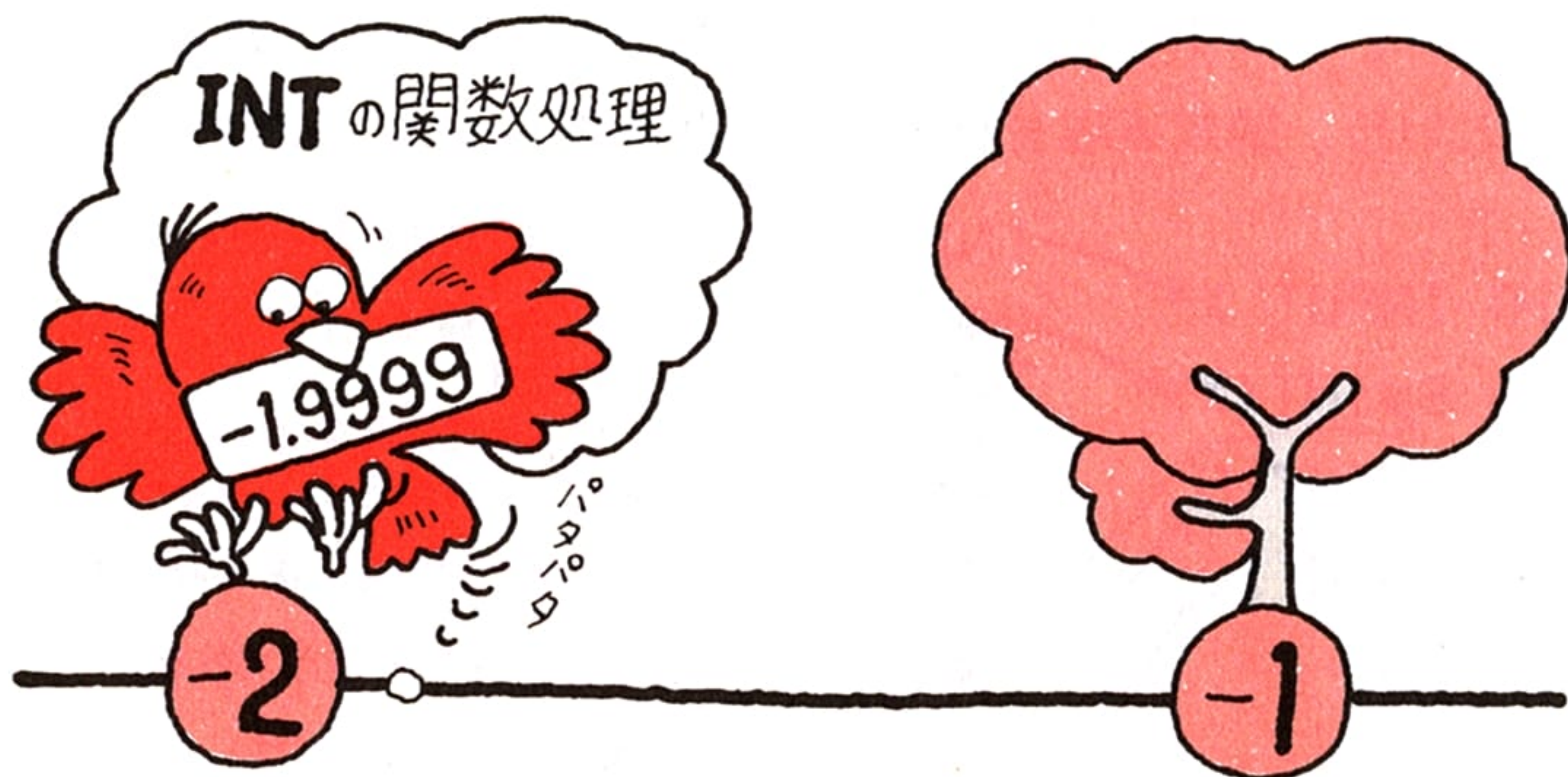
行番号20から送られたＳの中身が表示されます

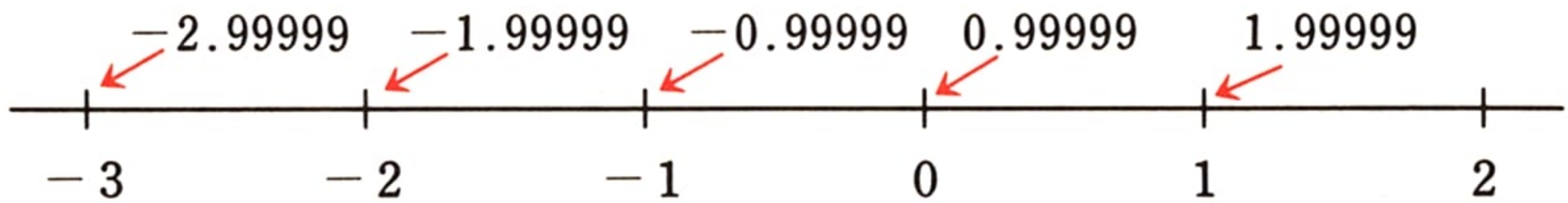
ＧＯＴＯ　10は、行番号10に飛べということです
そこで行番号10に飛び、行番号10を、また実行して、？マークを表示します

このようなプログラムにしたのは、ＩＮＴ(Ｘ)のＸに何個かの値を入れて、ＩＮＴ関数の処理をみるためです。

ＲＵＮを入力すると、行番号10のＩＮＰＵＴ　Ｘが実行されて、ディスプレイに？マークが表示されますから、小数点をともなった値を入力してみます。

これが、ＩＮＴの関数処理、つまり働きです。

この働きで、正の場合をみると、小数点以下を取り除く処理のようにみえますが、負の場合をみると、このようにいえないことがわかる

でしょう。これを、説明するのに一般に、“X以下でもっとも大きい整数”を返す、つまり算出するといいます。

このことは、下の図に示すことをいっているに過ぎません。

うえの図で、2.9999は2、1.9999は1、0.9999は0、−0.9999は−1、−1.9999は−2、−2.9999は−3といったように、すべて、左側にくる最初の整数をとっています。これが、INTの関数処理の仕方なのです。つまり2は2で返しますから、2.0001から2.9999までは、その左側にある一番最初の整数2をとって、返すということです。

また−1は−1で返しますから、−1.0001から−1.9999までは、その左側にある一番最初の値−2をとって、返すといった具合です。

このようなINTの働きから、INTはいろいろなことに使われます。

私たちが、INTをもっともよくみかけるのは、ゲームのプログラムのなかでしょう。ゲームのプログラムのなかでも、数あてゲームは、もっともポピュラーなものですから、このなかで、INTにお目にかかっているはずです。

数あてゲームのなかのINTは、つぎのようにRNDといっしょに使われています。

INT(RND(1)*10)

数あてゲームでは、INT(RND(1)*10)+1となっているはずですが、ここでは+1は、必要ないので省略します。

RND(X)は、(X)のXに、1とか2とかいった正の数を与えることによって、つぎのように乱数をつくりだします。

```
10 S=RND(1)
20 PRINT S
30 GOTO 10
40 END
```

```
run
 .421684
 .846581
 .681563
 .297739
```

つくりだされる数は、いろいろです。これは一例です。

ＲＮＤ（１）＊10の10は、ＲＮＤ（１）によってつくりだされた数を10倍するわけです。すると、うえの例の場合では、つぎのようになります。

```
run
 4.21684
 8.46581
 6.81563
 2.97739
```

しかし、このように単に10倍した数では、小数点以下をともなっているので、数あてゲームの数としては、複雑で不適当です。

そこで、ＩＮＴ（ＲＮＤ（１）＊10）として、ＩＮＴの働きをかりるわけです。

```
10 S=INT(RND(1)*10)
20 PRINT S
30 GOTO 10
40 END

run
 8
 4
 5
 3
 6
```

すると、ＲＮＤ（１）＊10でつくりだされた、小数点をともなった数は、ＩＮＴの〝Ｘ以下でもっとも大きな整数〟を返すという働きによって、うえの例のような整数になるのです。

サイコロの目のプログラム

```
10 PRINT "ｻｲｺﾛﾉﾒ"
20 PRINT
30 FOR I=1 TO 6
40 L=INT(RND(1)*6)+1
50 PRINT L;
60 NEXT I
70 PRINT
80 GOTO 30
90 END
```

INPUT

INPUTは、？マークを表示してデータを求めてくる

ＢＡＳＩＣでプログラムを書くとき、あらかじめプログラムのなかに、必要なデータを書いておく方法と、そうではなくて、パソコンがプログラムの実行に移ってから、必要なデータをキーボードから入力してやる方法とがあります。

ＩＮＰＵＴは、パソコンがプログラムの実行に移ってから、必要なデータを入力してやるものです。

つぎのプログラムは、ＩＮＰＵＴを使った簡単な足し算のプログラムです。

```
10 INPUT A,B    ← AとBの値を読み込みなさい
20 S=A+B        ← AとBの値を足して、その結果を左側のSに入れなさい
30 PRINT S      ← Sの値を表示しなさい
40 END          ← おわり
```

このプログラムでは、行番号10にＩＮＰＵＴが使われています。

```
10 INPUT A,B
```

ＩＮＰＵＴのあとにつづいているＡとかＢとかは、変数というものです。このようにＩＮＰＵＴのあとには、必ず変数がつづいています。これがＩＮＰＵＴの形です。

ところでＩＮＰＵＴ　Ａ，Ｂとは、どういう意味なのでしょうか。ＩＮＰＵＴ　Ａ，Ｂとは、ＩＮＰＵＴのあとにつづくＡとＢのデータ、つまり値を読み込みなさいという、パソコンに対する指示なのです。

しかし、まえに書いたようにＩＮＰＵＴは、パソコンがプログラムの実行に移ってから、必要とするデータを入力してやるものですから、プログラムのなかには、ＡとＢの値は書いてありません。

では、どうやってパソコンは、ＡとＢの値を読み込むのでしょうか。

まず、行番号10から40までのプログラムを全部、パソコンのなかに入力しましょう。プログラムを全部入力したら、ＲＵＮを入力します。ＲＵＮは、パソコンにプログラムを実行させる命令です。

するとパソコンは、一番最初の行番号10を実行します。

行番号10は、ＩＮＰＵＴ　Ａ，Ｂですから、それを実行するとパソコンは、ディスプレイに、？マークを表示します。

この？マークは、パソコンがＩＮＰＵＴ　Ａ，ＢのＡとＢの値はいくつなのか、私たちに尋ねているのです。つまり、ＡとＢの値をキーボードから入力するように、私たちに要求しているわけです。

パソコンは、私たちが？マークで要求している値を入力してやらないかぎり、つぎの行番号の実行に移りません。

そこで、AとBの値を入力してやりましょう。

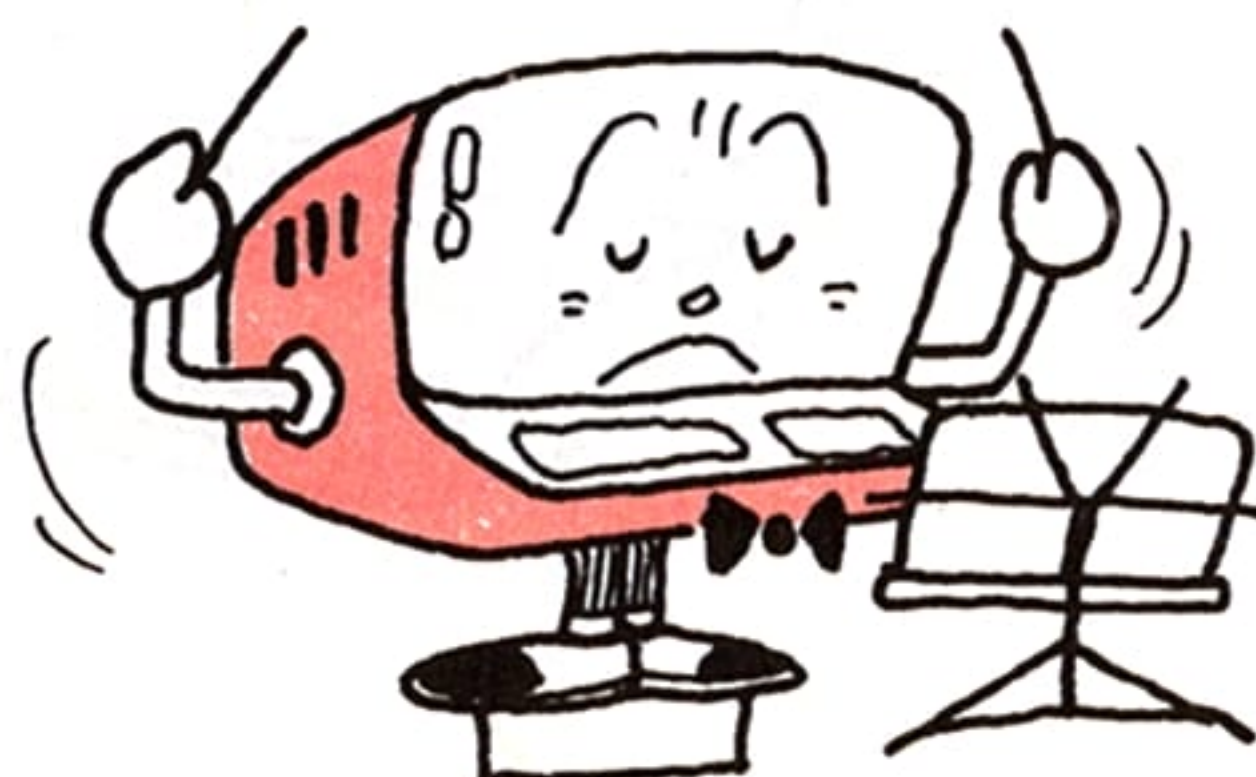
Aに入れる値を30、Bに入れる値を20とします。

Aに入れる値30を先に入力し、そしてつぎに「，」コンマを入力します。そのあとに、Bの値20を入力します。つぎのようにです。

そして、RETURNキーを押すと、パソコンは、30をAへ、20をBに読み込みます。

これが、INPUTを使ったときの、データの入力の仕方です。

では、INPUTで読み込まれたAとBの値は、どう処理されていくのでしょうか。データの流れ方を、順を追ってみてみましょう。

行番号10のINPUT　A、Bで読み込まれたAとBの値30と20は、つぎの行番号20のAとBに送られます。

行番号20のＡとＢに送られた値30と20は、計算式によって30＋20が行われます。30＋20＝50。この結果の50は、＝の左側にあるＳに入れられます。このＳに入れられた50は、つぎに行番号30のＰＲＩＮＴ　ＳのＳに送り込まれます。

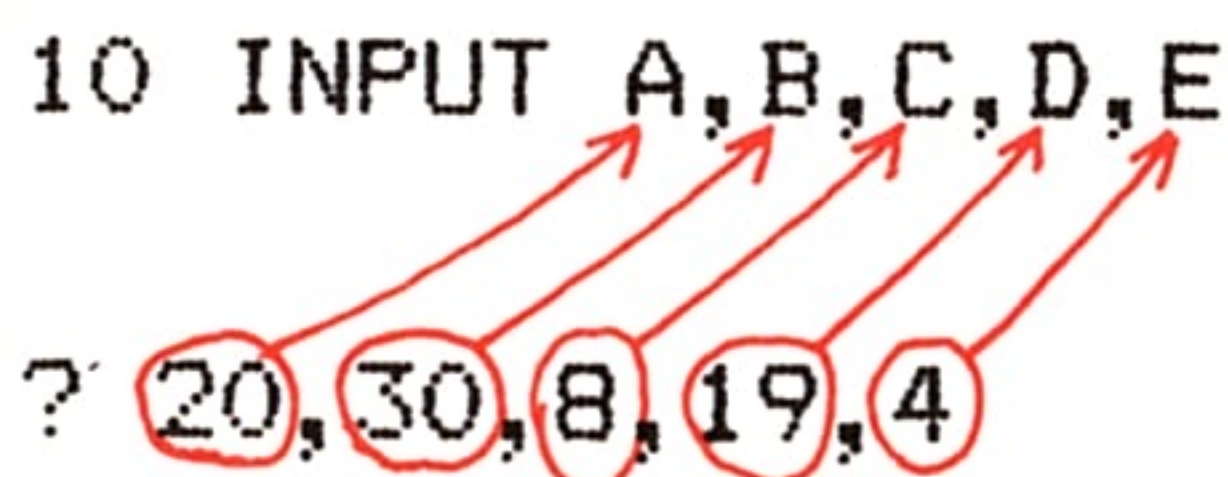
ＰＲＩＮＴとは、ディスプレイに表示しなさいということです。したがって、ＰＲＩＮＴ　Ｓによって、Ｓの値50がディスプレイに表示されます。

さて、まえにＡの値30とＢの値20を、「，」コンマで区切りました。これをコンマで区切るのを忘れたとしたら、どうなるでしょうか。

```
run
? 3020
??
```

RETURNキーを押します

すると？マークを２つ表示します

30と20の間をコンマで区切らないで入力し、ＲＥＴＵＲＮキーを押すと、パソコンは、今度は２つ？マークを表示してきます。これはパソコンが、不足しているＢの値を入力するよう、要求しているのです。なぜこうなったのかというと、30と20の間をコンマで区切るのを忘れてしまったために、パソコンは30と20を、3,020 というひとつの値とみなして、Ａに読み込んでしまったからです。そのため、Ｂの値が不足しているところから、パソコンは？マークを２つ表示して、Ｂの値

を入力するよう要求してきた、とこういうわけなのです。

このように、ＩＮＰＵＴのあとにつづく変数の個数に対して、入力したデータの個数がたりないと、パソコンは、変数の個数とデータの個数が一致するまで、？マークを表示して、データの入力を要求しつづけます。

パソコンに、データの個数を知らせるのは、「，」コンマですから、正確にコンマで区切って、変数の個数だけデータを入力してやらなければなりません。

また、入力したデータは、ＩＮＰＵＴにつづく変数に、最初から順に読み込まれていきます。

したがって、どの変数に、どの値を入れるのか、このことにも注意して入力してください。

ＩＮＰＵＴのかたち

ＩＮＰＵＴ　変数、変数、変数……………変数

？　定数、定数、定数……………定数

エラーメッセージ

パソコンはエラーメッセージでプログラムのミスを知らせる

パソコンのプログラムは、どんなに技術的に拙いものであっても、パソコンに実行させて、自分が望んでいる結果がえられれば、それで十分です。なぜならパソコンは、個人で所有し、個人で使うものだからです。

もちろん、パソコンにプログラムを入力して実行させても、最初からスムーズに、自分が望んでいる結果を得ることができるといったことは、まれでしょう。

プログラムの実行を命令するＲＵＮを入力したあと、一度か二度は、必ずといっていいほどパソコンに、プログラムの間違いを知らされるものです。

このようにパソコンがプログラムに間違いがあることを発見して、私たちに伝えてくれることばを、エラーメッセージといいます。

エラーメッセージのなかで、もっとも知られているメッセージが、Syntax error でしょう。

エラーメッセージ

Syntax error は、文法が間違っているときとか、プログラムのなかに、決められた以外のステートメントがあるときに、知らせてくれるメッセージです。

```
10 INPUT A,B
20 PRYNT A,B
30 END
run
? 20,30
Syntax error in 20
```

Yではなくて I

行番号20に文法の間違いがあるためのエラーメッセージ

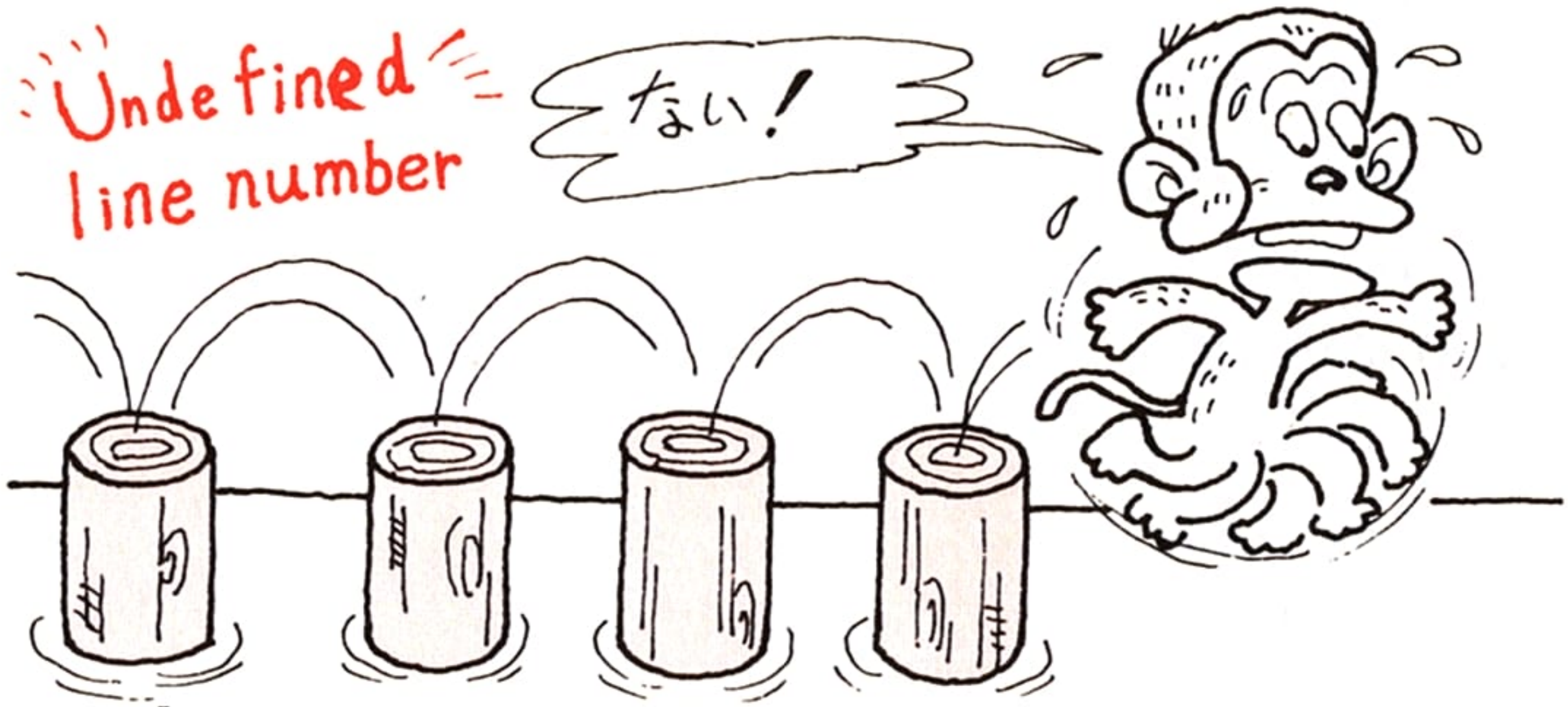
また、よく表示されるエラーメッセージとしては、つぎのようなものが、あげられるでしょう。

```
10 FOR I=1 TO 10
20 PRINT I
30 NEXT T
40 END
run
NEXT without FOR in 30
```

変数を I なら I に一致させなければなりません

変数が一致していないためのエラーメッセージ

NEXT without FORというエラーメッセージの意味は、ＦＯＲ～ＮＥＸＴが正しく対応していない、というものです。

うえのプログラムでは、行番号10のＦＯＲでは、Ｉになっているに

もかかわらず、行番号30のＮＥＸＴでは、Ｔになっているためのエラーメッセージです。入力ミスによって、このようなことがよく起こります。

```
10 READ A
20 S=A*A
30 PRINT S
40 GOTO 70
50 DATA 1,2,3,4,5
60 END
run
Undefined line number in 40
```

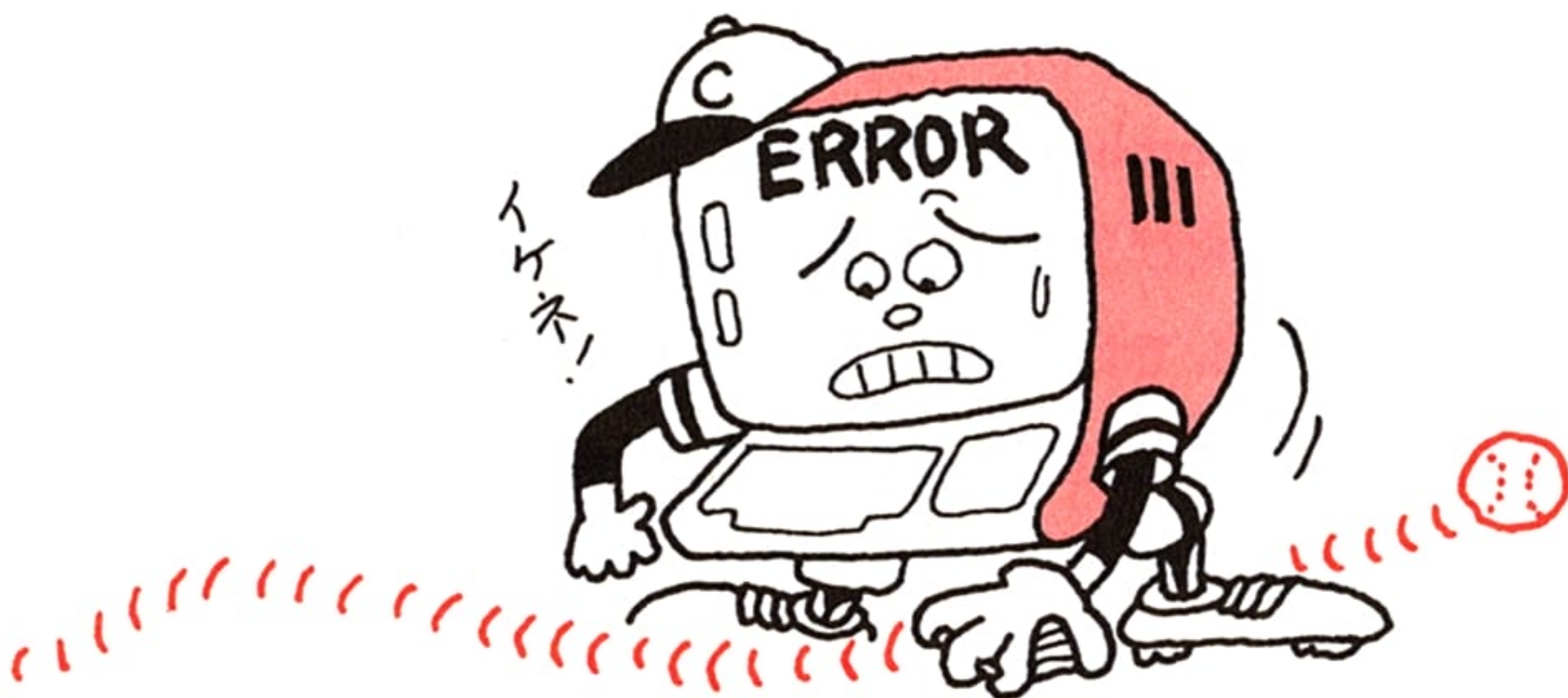

Undefined line numberというエラーメッセージは、必要とされる行番号（ＧＯＴＯの飛び先など）が定義されていない、というものです。

うえのプログラムでは、40　ＧＯＴＯ　10として、行番号10のＲＥＡＤ　Ａにジャンプさせて、繰り返し行番号50のＤＡＴＡの値を読みとらせるべきなのですが、プログラムにない行番号にジャンプさせています。このようなことも、入力ミスによってよく起きます。

```
10 READ A,B
20 GOSUB 100
30 READ A,B
40 GOSUB 100
100 C=A*B
110 PRINT C
120 RETURN
200 DATA 2,3
210 DATA 4,5
run
 6
 20
 20
RETURN without GOSUB in 120
```

メインルーチンの最後にENDがありません。50　ENDを入れないと、エラーメッセージが表示されます

メインルーチンの終りにENDがないためのエラーメッセージ

RETURN without GOSUBというエラーメッセージは、GOSUB～RETURNが正しく対応していない、というものです。このプログラムでは、メインルーチンの最後にENDがないために表示された、エラーメッセージです。行番号40のあとに、50　ENDを入れると、エラーメッセージは表示されません。

少しむずかしくなりますが、つぎのようなプログラムの場合に、Out of DATAというエラーメッセージが表示されます。

Out of DATAとは、READ文で読まれるべきデータが、DATA文のなかに用意されていない、というものです。

```
10 DIM A(30)
20 FOR I=1 TO 30
30 READ A(I)
40 NEXT I

180 DATA 1,5,20,13,6,19,7,18,23
190 DATA 4,16,11,9,22,12,17,2,21
200 DATA 25,14,10,15,19,8,20,24,3
run
 Out of DATA in 30
```

READ A(I)はFOR～NEXTによって30個のデータを読みとろうとしますが、データの数が27個しかありません

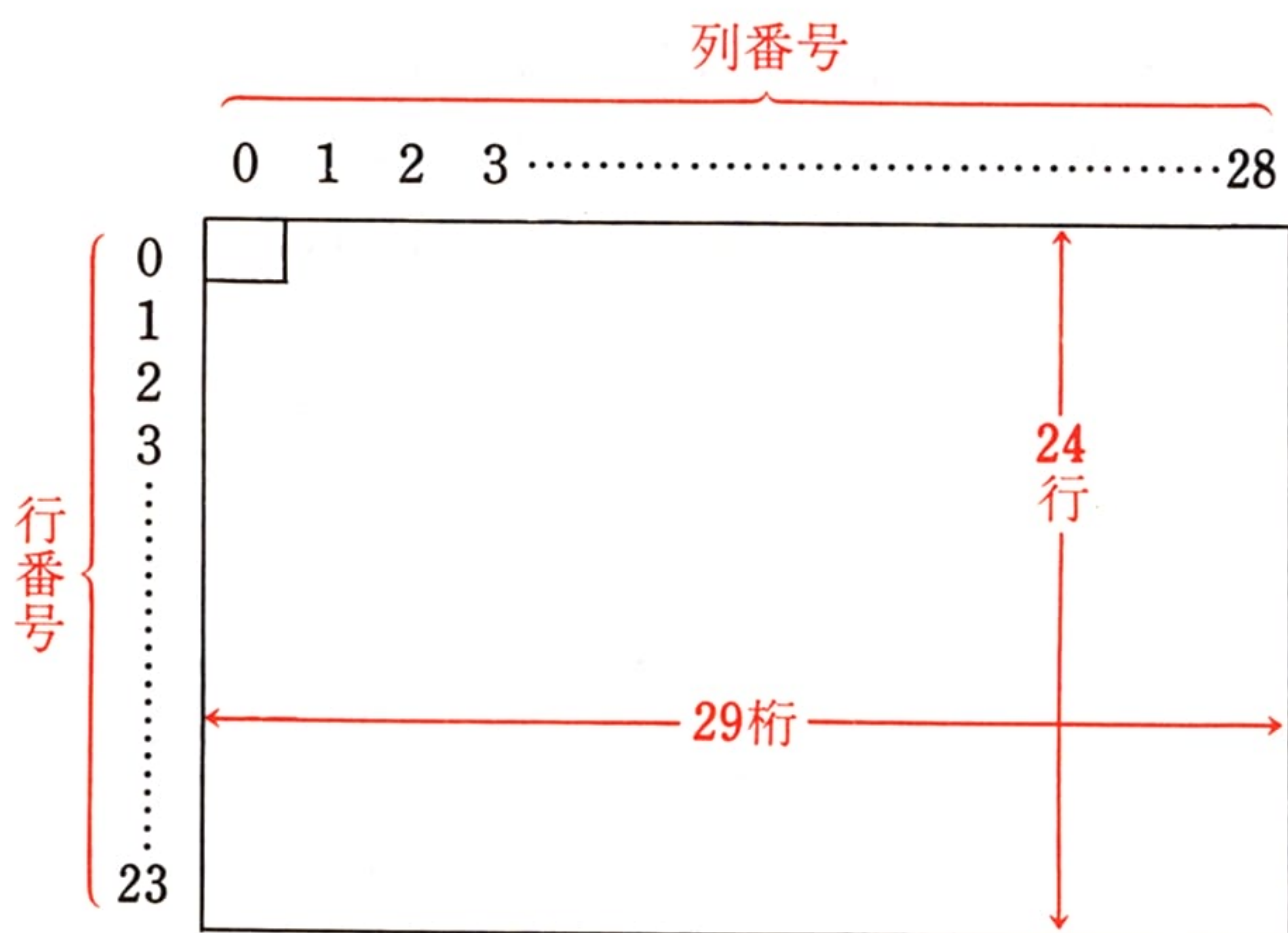
READ文で読みとるデータが、DATA文の中に不足しているためのエラーメッセージ

このプログラムでは、FOR　I＝1　TO　30になっています。したがって、行番号40のNEXT　Iと対応して、Iが1から30になるまで、行番号30のREAD　A（I）に、DATAに示されているデータを読みとらせようとします。しかし、DATAに示されているデータは、27個しかありません。つまり3個不足しています。そのために、エラーメッセージが表示されたわけです。

WIDTH

WIDTHは、画面に表示できる文字数を変える

パソコンをスイッチオンしたとき、ディスプレイの画面に表示できる文字の数は、横１行に29桁、行数は24行となっています。下の図が、それです。

これはスイッチオンしたとき、自動的にセットされます。

この各桁、各行には、０からの番号がつけられていて、これを列番号、行番号と呼んでいます。この番号は、画面表示に関する命令を使うときに必要ですので、おぼえておいてください。

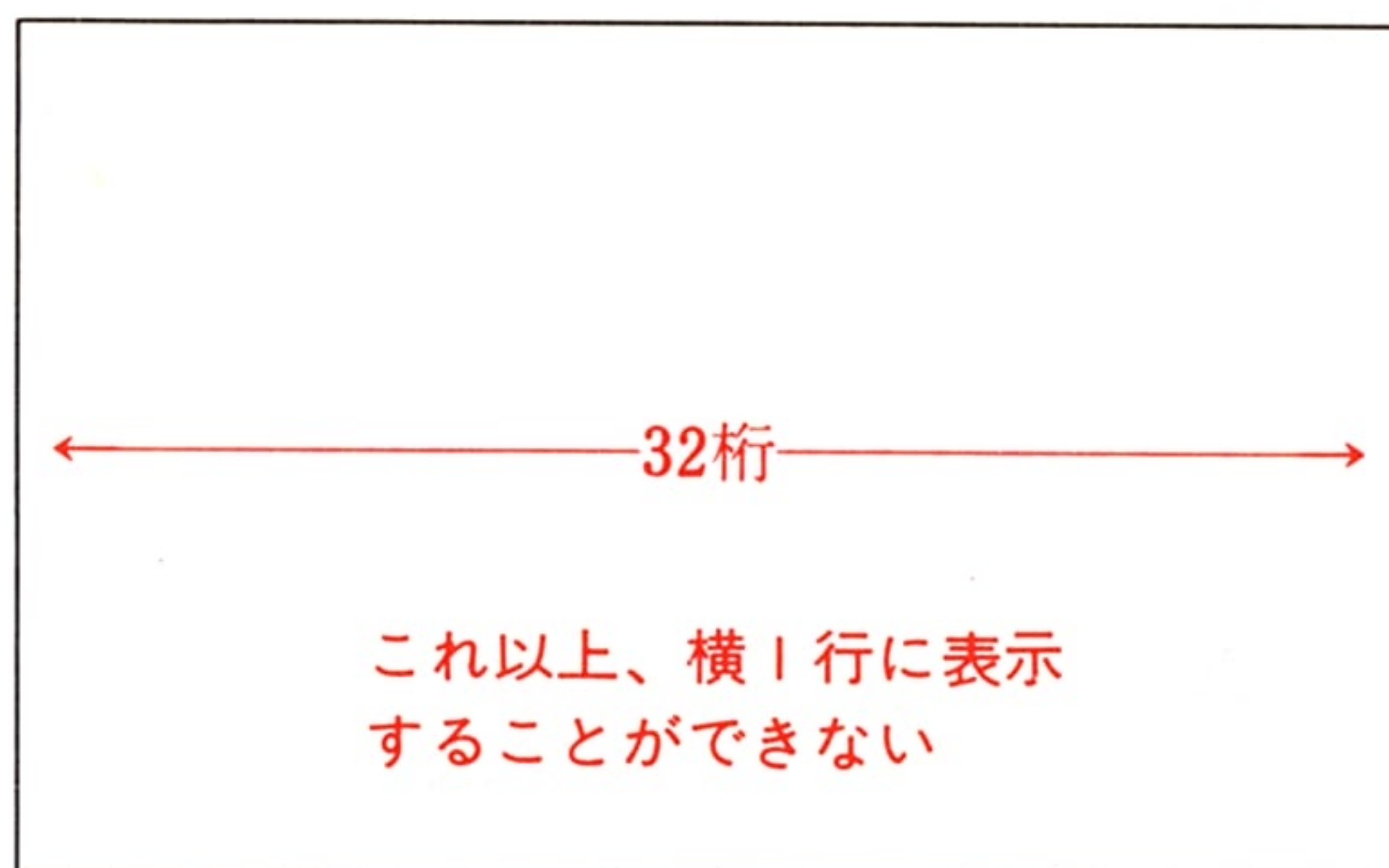
ところでＷＩＤＴＨは、ディスプレイの画面に表示できる、横１行の文字の桁数を変えるときに使います。

たとえば、つぎのように、

```
10 WIDTH 32
```

文字の桁数の指定
こうすると、1行に32文字入るようになります

とすると、パソコンにスイッチオンしたとき、横1行に29桁しか表示することができなかったものが、横1行に32桁表示することができるようになりました。

ただし、この横1行32桁が、ディスプレイに表示することができる最大の桁数となっていますから、これ以上の桁数を1行に表示させることはできません。

このように、ＷＩＤＴＨで、横１行に表示する桁数を32桁に変更すると、列番号はそれにともなって、まえの図に示したように０から31にかわります。ＷＩＤＴＨで変更することができるのは、桁数だけとなっています。行数は変更することができません。行数は、24行に固定されています。

ところでＷＩＤＴＨで、横１行に表示する桁数を変更すると、いつまでも変更したままの状態が続きます。たとえば、まえのように、ＷＩＤＴＨで横１行32桁表示に変更すると、

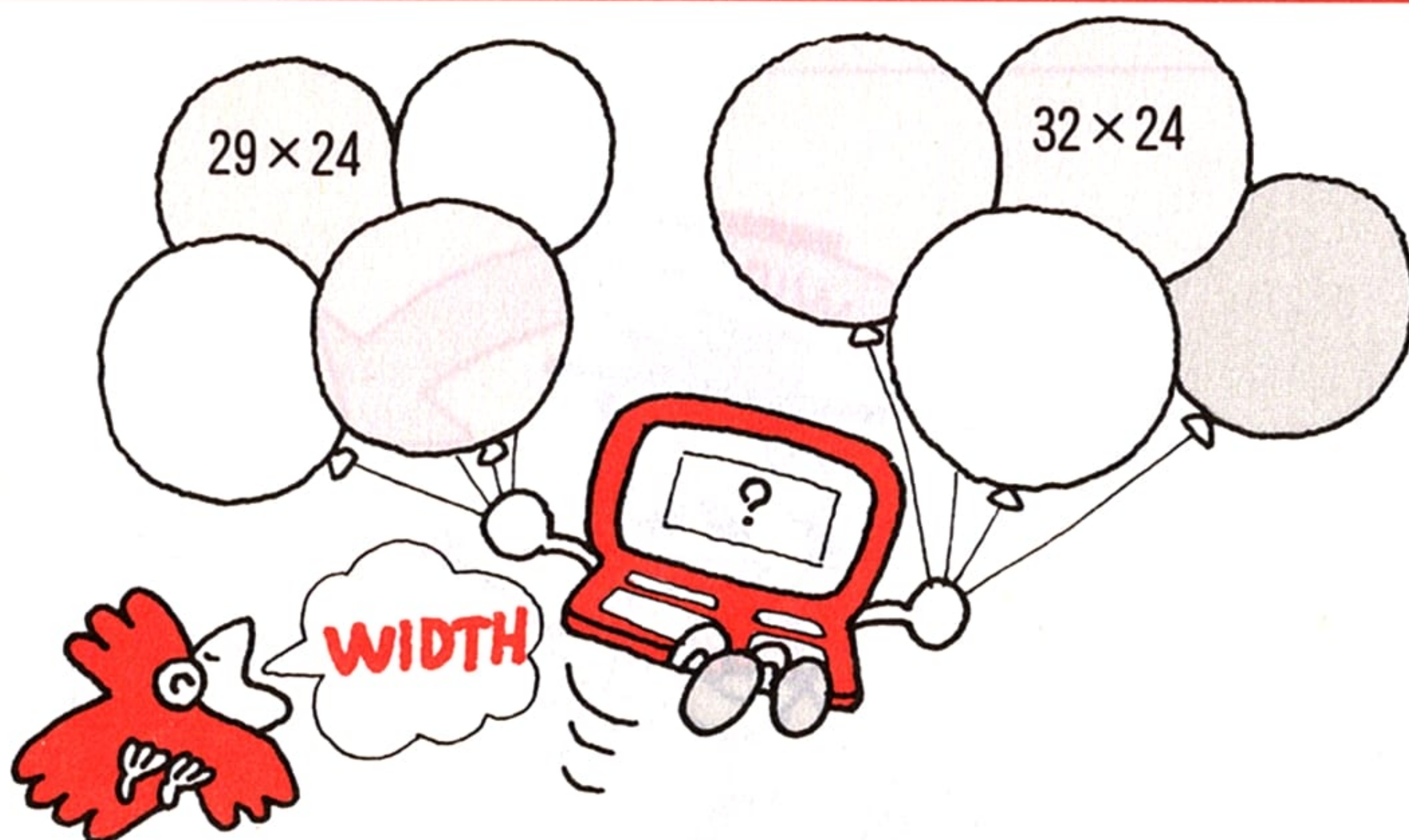
WIDTH　32

いつまでも、横１行32桁表示となっています。これをパソコンに電源を入れたときの横１行29桁表示に戻すには、もういちど、ＷＩＤＴＨを使って指定します。

WIDTH　29

このようにすると、もとの横１行29桁表示に戻ります。

ＷＩＤＴＨのかたち

ＷＩＤＴＨ　<桁数>

END

ENDは、パソコンにプログラムのおわりを知らせる

ＥＮＤは、プログラムの最後につけます。このＥＮＤは、パソコンに〝これでプログラムはおわりです〟ということを知らせるもので、パソコンは、このＥＮＤにであうと、これでプログラムはおわりだと判断して、プログラムの実行を終了したというサインである、ＯＫを表示します。

```
10 A=5
20 B=8
30 C=A/B
40 PRINT C
50 END

run
.625
Ok
```

- 10 A=5 ← 5をAに入れなさい
- 20 B=8 ← 8をBに入れなさい
- 30 C=A/B ← C＝A／BのAとBには、行番号10のAの値と行番号20のBの値が送り込まれます。そして5÷8が行なわれ、その結果の 0.625が左側のCに入ります
- 40 PRINT C ← PRINT　CのCには、行番号30のCから 0.625が送り込まれます。そしてPRINT　Cで 0.625が表示されます
- 50 END ← おわり
- .625 ← 行番号30の計算結果です
- Ok ← 行番号50のENDで、OKが表示されます

ＥＮＤにつける行番号は、うえのプログラムでは、40のつぎの50にしましたが、必ずしもこのようにつづける必要はありません。つぎのようにしても、よいのです。

```
40 PRINT C
90 END
```

← このようにしてもかまいません

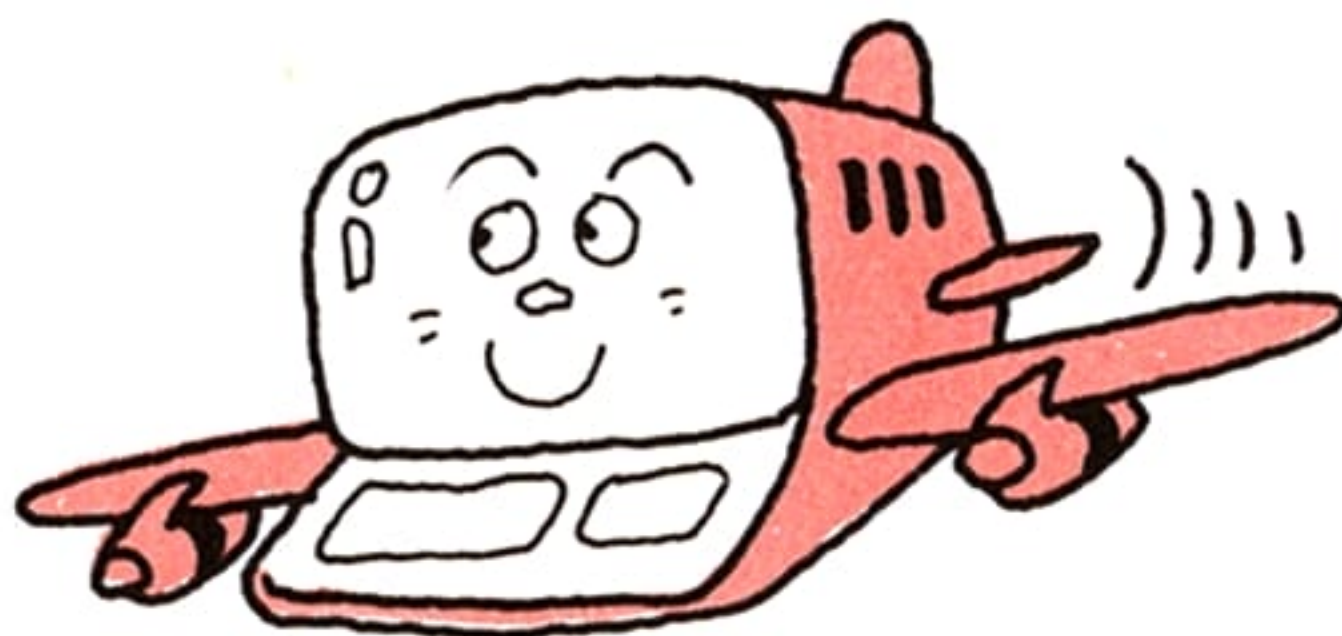
今日では、ほとんどのパソコンで、このＥＮＤを省略することができますが、すべての場合、このＥＮＤが省略できるわけではありません。

たとえば、サブルーチンをともなったプログラムでは、メインルーチンのおわりに、必ずＥＮＤをつけなければなりません。そうしないとパソコンは、

ＲＥＴＵＲＮ　without　ＧＯＳＵＢ

（ＧＯＳＵＢとＲＥＴＵＲＮが対応していない。ＲＥＴＵＲＮだけがある）

といったエラーメッセージを表示します。

このように、必ずＥＮＤをつけなければならないこともあるので、たとえＥＮＤを省略することができるとしても、プログラムのおわりには、ＥＮＤをつけるようにしておいたほうがよいでしょう。

AUTO

AUTOは、パソコンに、行番号を自動的につけさせる

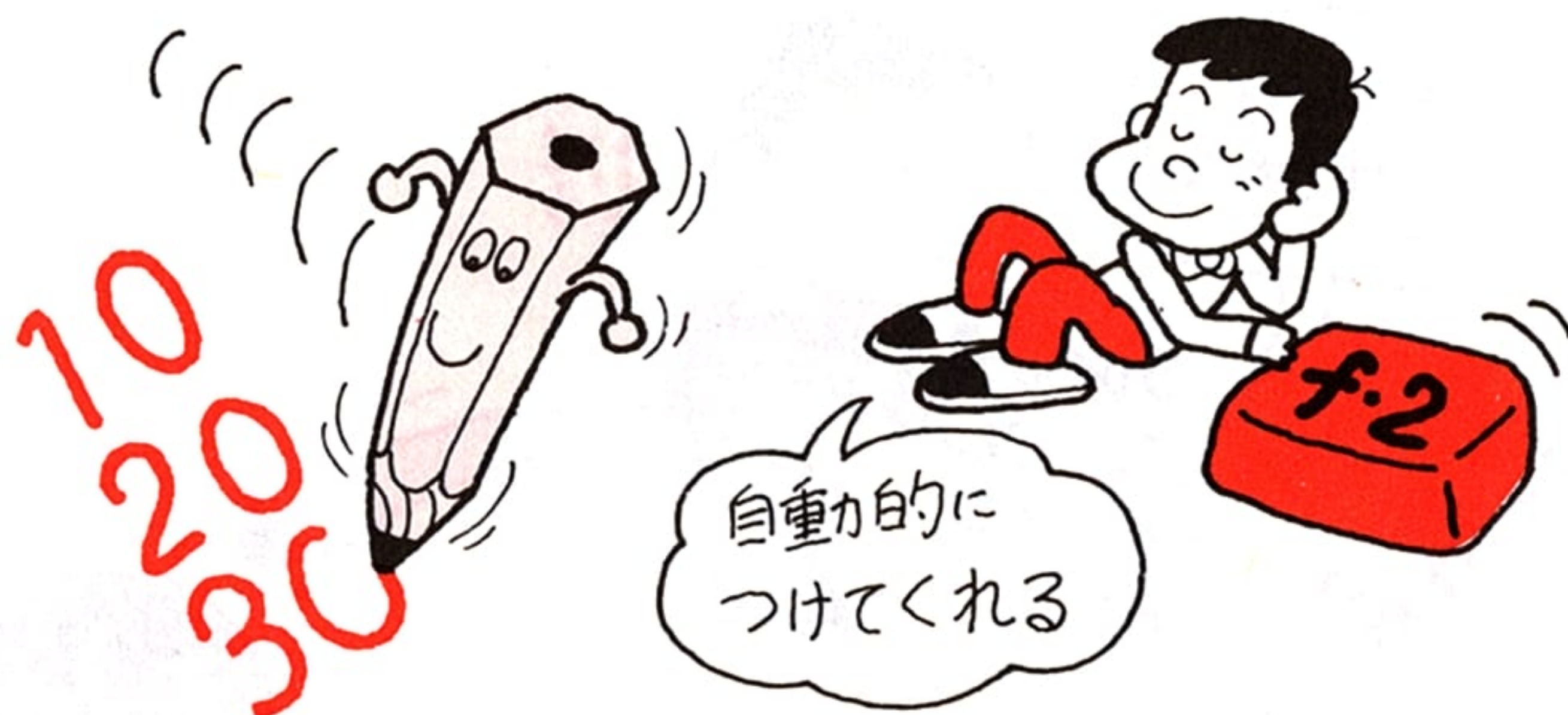
私たちが、パソコンにプログラムを入力するとき、まず行番号を入力し、ついでステートメントを入力します。

ステートメントは、プログラムによってすべて異なりますが、行番号はどんなプログラムでも、順番についている、決まりきったものです。これをパソコンが、自動的につけてくれるだけで、プログラムの入力は、いちだんと楽になります。

このことをパソコンに行わせる命令が、ＡＵＴＯです。

ふつうパソコンはファンクションキーのf･２が、ＡＵＴＯコマンドのキーです。コマンドとは、命令という意味で、私たちがパソコンに直接与える命令を、そう呼んでいます。ＲＵＮなどと同じです。

さて、プログラムを入力するまえに、f･２キーを押します。

すると、ＡＵＴＯとディスプレイに表示されます。つづいてＲＥＴ

URNキーを押すと、改行されて10が表示されます。

```
auto
10
```

auto ← 行番号を自動的に発生させるコマンド
10 ← 行番号

この10が一番最初の行番号です。この場合は10ごとに、自動的に行番号がつけられていきます。

```
auto
10 INPUT A,B
20 S=A*B
30 PRINT S
40 END
```

AUTOのあとに何も指定しないと、10ごとに行番号が自動的につけられる

最後のプログラムを入力してRETURNキーを押したら、AUTOを解除するSTOPキーとCTRLキーを一緒に押します。

この場合、40　ENDが最後ですから、これを入力して、RETURNキーを押すと、50が表示されます。ここでSTOPキーとCTRLキーを一緒に押します。そうすると、AUTOが解除されます。

STOPキーとCTRLキーを一緒に押さないとAUTOが解除されないので、いつまでも行番号を発生しつづけることになります。

では、行番号を10からはじめて、20づつ増やしてつけるときは、どうしたらよいでしょうか。

その場合は、まず、f･2 キーを押します。するとＡＵＴＯと表示されますから、その横に10、20と入力します。つぎのようにです。

```
auto 10,20
```

ＲＥＴＵＲＮキーを押すと、一番最初の行番号10が表示されますが、そのあとは、20づつ増えて、行番号が自動的につけられていきます。

```
auto 10,20
10 READ A,B
30 C=A*B/2
50 PRINT C
70 DATA 30,20,50,8,19,25
90 END
```

行番号10から、20づつ増えて
行番号が自動的つけられる

最後に100からはじめて、10ごとに行番号を自動的につけていく例をあげておきましょう。つぎのようにすると、100からはじまって、10ごとに行番号が自動的につけられていきます。

```
auto 100,10
100 A=10
110 B=20
120 C=A*B/2
130 PRINT C
140 END
```

行番号 100から10ごとに
行番号が自動的につけられる

AUTOのかたち

ＡＵＴＯ　最初の行番号を指定、増加させる数を指定

カーソル

カーソルは、プログラムの訂正にも欠かせない

パソコンにスイッチオンすると、ディスプレイの画面上に文字が表示されます。その文字の下のほうに、文字より少し大きめの四角い印（■）が表示されます。

キーボードのキーを叩くと、四角い印が表示されていたところに、叩いたキーの文字が表示されます。そして四角い印は、表示された文字のひとつ右側に移動します。

この四角い印が、カーソルといわれるものです。

カーソルは、キーボードのキーを叩いたとき、ディスプレイの画面上に、叩かれたキーの文字や記号が表示される位置を、私たちに知らせてくれているのです。

キーのうえにある［HOME　CLS］というキーと［SHIFT］キーを一緒に押すと、カーソルは画面の一番うえ、左端に移動します。ここがカーソルの定位置（ホームポジション）です。つまり、文字や記号などを表示するはじまりの位置ということです。

この位置から、キーボードのキーから入力された文字や記号などが、表示されていきます。そして、文字や記号などが表示されるたびに、カーソルは右へ、ひとつづつ移動していきます。ディスプレイの画面の右端までくると、カーソルは、画面の左端、つまり、つぎの行の先頭の位置に移ります。

これまでおはなししたように、カーソルと文字や記号などの表示は、

切っても切れない関係にあるのです。

ということは、カーソルは、プログラムの訂正にも大きくかかわってくるはずです。なぜなら、カーソルのある位置にしか文字や記号を表示できない以上、カーソルを訂正するところまで移動して、正しい文字や記号に換えなければならないからです。

カーソルを移動させるには、矢印のキーを使います。キーの右側にある４つのキー（←↑↓→）です。

上方向（↑）を示している矢印のキーを一度押すと、カーソルは画面の上に向かって１行分移動します。キーを押し続けると、カーソルは画面の上に向かって続けて移動します。下方向（↓）を示している矢印のキーは、その逆の働きです。

左の方向（←）を示している矢印のキーを一度押すと、カーソルは画面の左に向かって、１桁分（１字分）移動します。キーを押し続けると、カーソルは画面の左に向かって続けて移動します。右方向(→)を示している矢印のキーは、その逆の働きです。

このように、矢印のキーを使ってカーソルを移動させます。プログラムの訂正は、カーソルを訂正する箇所まで矢印のキーで移動させて、正しい文字や記号をキーボードから入力して、行います。

キャラクタ・コード

文字や数字、記号には10進数のコードがつけられている

私たちが書くBASICのプログラムは、つぎのプログラムのように、アルファベット、カタカナ、数字、記号などが、いろいろに使われています。

```
10 READ A,B,C
20 S=A+B-C
30 PRINT A;"タス";B;"ヒク";C;"=";S
40 DATA 9,8,6
50 END
```

私たちは、このプログラムを、そのとおりにキーボードのキーを叩いて入力しますが、AとかB、＋とか－、タとかス、9とか8といったアルファベットやカタカナ、数字、記号などが、そのままの形で、パソコンのなかに入っていくわけではありません。

これらはAは65、Bは66、＋は43、－は45、タは192、スは189、9は57、8は56といったように、いったん10進数のコードに置き換えられます。そしてつぎに、その10進数のコード、つまりAの65は01000001、Bの66は01000010、＋の43は00101011、－の43は00101101、タの192は11000000、スの189は10111101、9の57は00111001、8の56は00111000といったように、8ビットの2進数に変換されて、パソコンのなかに入っていきます。

この場合のAは65、Bは66、＋は43、－は45、タは192、スは189、9は57、8は56といった10進数のコードが、キャラクタ・コードとい

うものです。

キャラクタとは、BASICで使うアルファベットやカタカナ、数字、記号などの総称で、キャラクタ・コードは、これらすべてのものにつけられています。

このキャラクタ・コードを知るには、キャラクタ・コード表をみてもわかりますが、キャラクタ・コード表は、16進数で表わされていることが多いので、10進数でキャラクタ・コードを知りたいときは、つぎのように、ASC（X$）を使うと簡単に知ることができます。

```
10 PRINT ASC("カ")
20 PRINT ASC("M")
run
 182
 77
```

ASC（X$）については、12頁を参照してください。

またCHR$（X）を使うと、この反対にキャラクタ・コードを、そのコードに該当するキャラクタに変換します。

```
10 PRINT CHR$(184)
20 PRINT CHR$(67)
run
ク
C
```

CHR$（X）については、82頁を参照してください。

行番号

パソコンは、行番号の小さいほうから順に整理し、記憶する

ＢＡＳＩＣで書いた文をステートメントといいます。プログラムは、このステートメントが集まってできています。

```
10 A=5        ← 5をAに入れなさい
20 B=2        ← 2をBに入れなさい
30 C=A+B      ← Aの中身5とBの中身2をたして、
                その結果の7を左側のCに入れなさい
40 PRINT C    ← Cの中身7を表示しなさい
50 END        ← おわり
```

このプログラムの〝Ａ＝５〟や〝Ｃ＝Ａ＋Ｂ〟、〝ＰＲＩＮＴ　Ｃ〟などが文、つまりステートメントです。

このようにステートメントは、ふつう１行にひとつ書いて、その頭には、必ず番号をつけます。これを、行番号といいます。うえのプログラムの場合では、左側に並んでいる10から50までの番号が、それです。

この行番号は、なんのために必要なのでしょうか。

パソコンは、この行番号の小さいものから大きいものへと、順番に読みとっていき、そして整理して記憶するのです。

たとえば、まえのプログラムを、つぎのようにして入力してみてください。

```
20 B=2
10 A=5
40 PRINT C
50 END
30 C=A+B
```

たとえこのようにプログラムを入力したとしても、パソコンは、行番号の小さいものから大きいものへと、順番に整理して記憶しているのです。

もし、このことを確かめるには、［HOME　CLS］キーと［SHIFT］キーを押して画面を消し、ＬＩＳＴというコマンド（命令）を入力してみてください。ＬＩＳＴというコマンドは、メモリのなかに記憶されているプログラムを、ディスプレイのうえに表示させるものです。

ディスプレイに表示されたプログラムは、10、20、30、40、50といった具合に、小さいものから順序よく並んでいるはずです。

また、パソコンは、この行番号の小さいものから大きいものへと順番にステートメントを実行していきます。

このように行番号は、単なる番号ではなく、パソコンにとっては、プログラムを記憶したり、プログラムを実行したりするための手順となる、大切なものなのです。

ところで、まえに示したプログラムの行番号のように、行番号は、10、20、30、40、50といった具合に、10ごとにつけられています。これは、１、２、３、４、５といったように、連続した番号にしてもよ

いはずです。

ただ、1、2、3、4、5といった連続した番号にしないのは、プログラムを作ったあとで、書き忘れたステートメントがあったり、つけ加えたいステートメントがでてきたときに困るからです。

たとえば、つぎのようなプログラムの行番号30と40、行番号40と50の間に、ステートメントを2つ追加したいとします。

```
10 A=3
20 B=5
30 C=A+B
40 PRINT C
50 END
35 D=A/B ←――― 行番号30と40の間に追加したいステートメント
45 PRINT D ←――― 行番号40と50の間に追加したいステートメント
```

このようなとき、10ごとに行番号がつけられていると、31から39、また41から49までの番号のなかから任意に番号をえらんで、追加したいステートメントの頭につけて、50　ENDのあとにつけ加えておけばすみます。

ところが、1、2、3、4、5といったように連続した行番号をつけておくと、このような具合にはいきません。なぜなら、行番号は整

数でなければならないと決まっていて、3.5とか、4.5とかいった小数点をともなった行番号をつけることができないからです。

したがって、1、2、3、4、5といった連続した行番号をつけていると、このように追加したいステートメントが生じたとき、行番号を全部つけ直すといった、はめになります。

このようなことがないよう、ふつうは、行番号を10ごとにつけているというわけなのです。

サイコロゲームのプログラム

```
5   PRINT CHR$(12)
10  LOCATE 13,7
20  PRINT "ｻｲｺﾛｹﾞｰﾑｦﾔﾛｳ"
30  FOR I=1 TO 1500
40  NEXT I
45  PRINT CHR$(12)
50  S=1000
60  PRINT "ｷﾐﾉﾓﾁﾃﾝﾊ";S;"ﾃﾞｽ"
70  PRINT "ｲｸﾗｶｹﾏｽｶ";
80  INPUT B
90  PRINT "1ｶ0ｲﾚﾃｸﾀﾞｻｲ";
100 INPUT L
110 X=INT(6*RND(1))+1
120 Y=INT(6*RND(1))+1
130 E=X+Y-2*INT((X+Y)/2)
140 PRINT
150 PRINTX;Y;
160 IF E=1 THEN 220
170 PRINT "  ﾅﾉﾃﾞ0ｶﾞｱﾀﾘ"
180 FOR I=1 TO 2000
190 NEXT I
200 PRINT CHR$(12)
210 GOTO 255
220 PRINT "  ﾅﾉﾃﾞ1ｶﾞｱﾀﾘ"
230 FOR I=1 TO 2000
240 NEXT I
250 PRINT CHR$(12)
255 PRINT
260 IF L=E THEN 290
270 S=S-B
280 GOTO 60
290 S=S+INT(.8*B)
300 GOTO 60
310 END
```

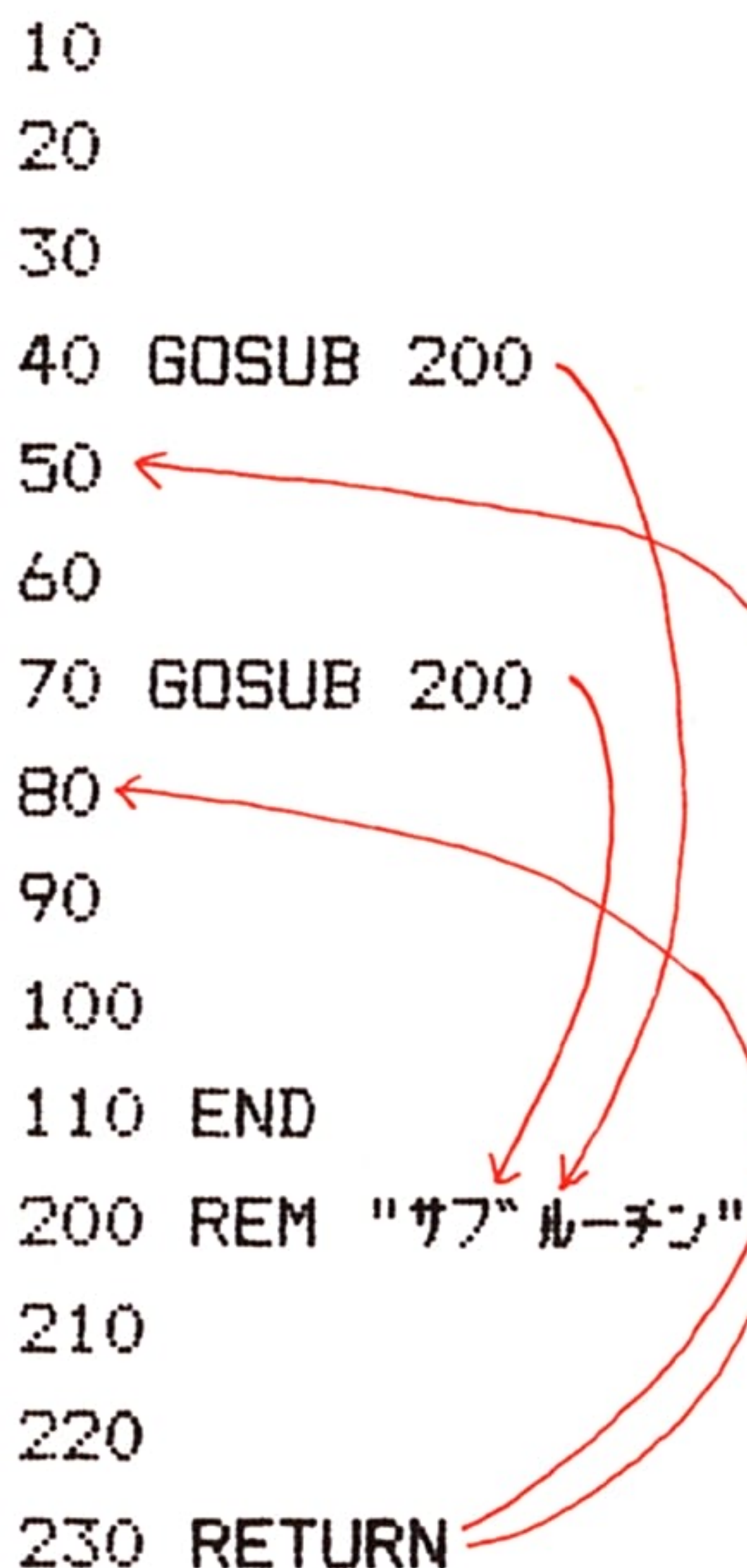
GOSUB〜RETURN

GOSUBは、サブルーチンへ飛べ、RETURNは、戻れ

プログラムを作っていると、同じ計算などが何回もくり返してでてくることが、しょっちゅうあります。

同じ計算がでてくるたびに、いちいち書いていたのではめんどうですし、第一、プログラムが複雑になって、書き間違えることが多くなって困ります。

メインルーチンとサブルーチン

このようなとき、くり返しでてくる計算などをひとまとめにしてしまって、1度書けばすむようにできれば、非常に便利です。

この方法が、サブルーチンです。サブルーチンとは、副プログラムということです。

つまり、サブルーチンは、何回もでてくる計算などをひとまとめにしてしまって、それを本筋のプログラムから切り離してしまい、副プログラムにしてしまうことです。

この場合、本筋のプログラムをメインルーチン（主プログラム）といいます。

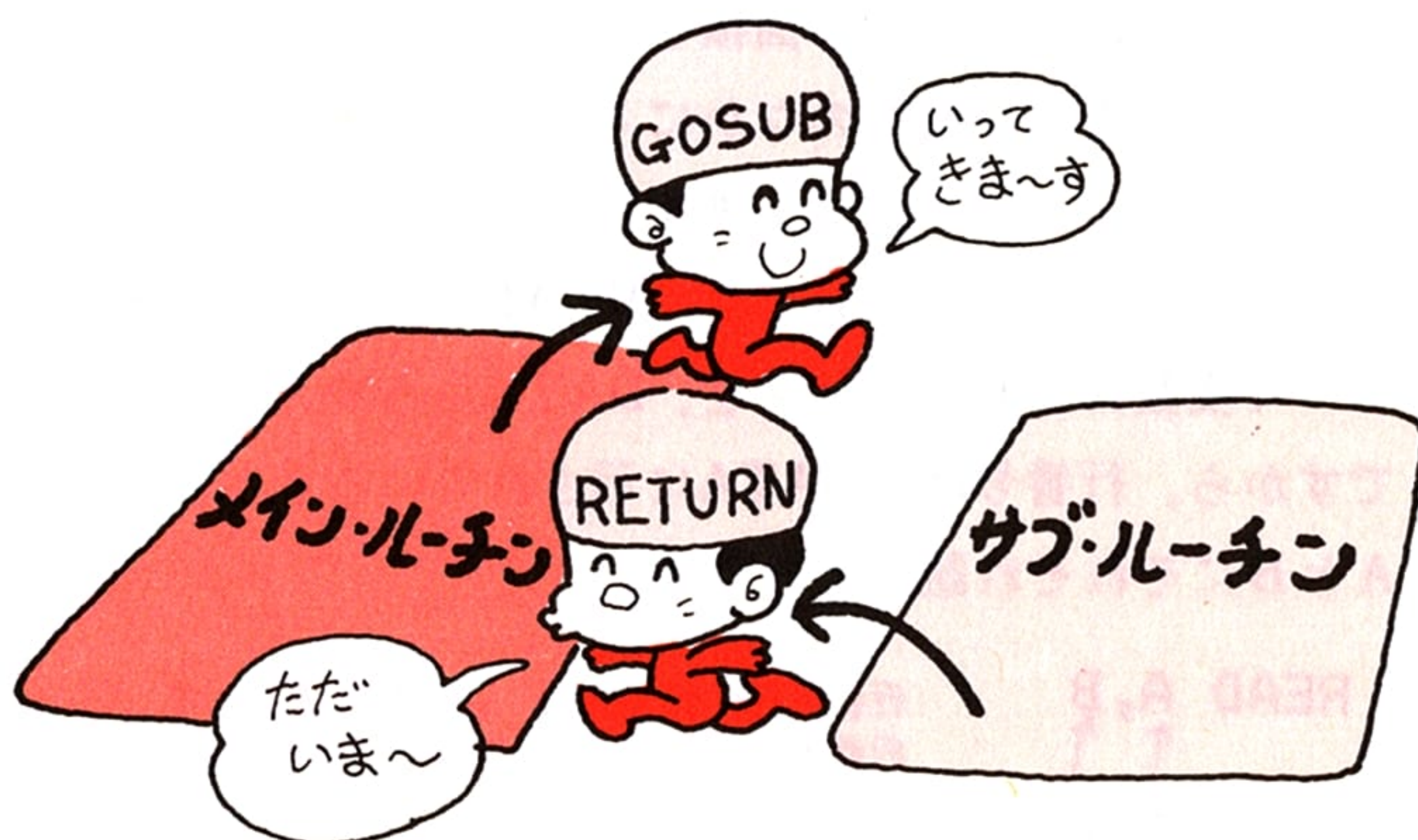
メインルーチンから切り離したサ

ブルーチンは、前頁の左の図のように、メインルーチンのおわりのEND文のあとにつづけます。

そしてメインルーチンで、サブルーチンにした計算などが必要となったところで、サブルーチンへ飛ばして計算などをさせ、ふたたびメインルーチンに戻して、メインルーチンを実行させるようにするわけです。

ところで、プログラムのなかにサブルーチンを作ったとき、どうしても必要になる言葉があります。それが、GOSUB～RETURNです。GOSUBは、サブルーチンへ飛べ、RETURNは、メインルーチンへ戻れという意味です。

したがって、GOSUBのあとには、飛ぶべきサブルーチンの一番最初の行番号がつづきます。

RETURNのあとには、何もいりません。これは、パソコンの内部で戻るべき行番号を記憶していて、自動的に、サブルーチンに飛んだ次の行番号に戻るようになっているからです。

では、簡単なプログラムで、GOSUB～RETURNの働きをみてみましょう。はじめのプログラムは、GOSUB～RETURN を使っていません。次のプログラムが、最初のプログラムを、GOSUB～RETURNを使って書きなおしたものです。

```
10 READ A,B
20 C=A*B/2
30 PRINT "ｻﾝｶｸｹｲﾉﾒﾝｾｷ";C
40 READ A,B
50 C=A*B/2
60 PRINT "ｻﾝｶｸｹｲﾉﾒﾝｾｷ";C
70 END
80 DATA 8,15
90 DATA 15,20
run
ｻﾝｶｸｹｲﾉﾒﾝｾｷ 60
ｻﾝｶｸｹｲﾉﾒﾝｾｷ 150
```

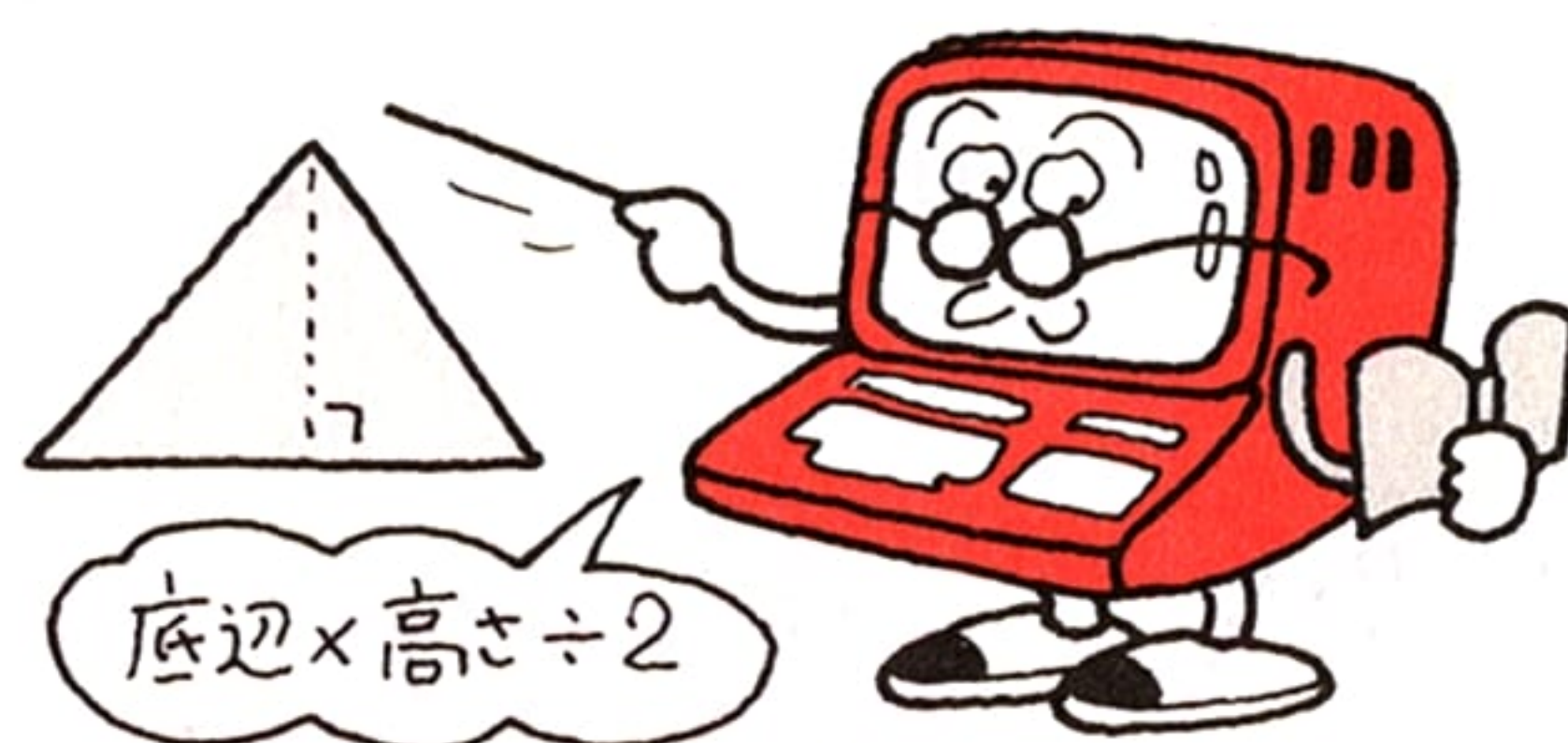
このプログラムは、三角形の面積を2つ求めるプログラムです。

このプログラムをパソコンのなかに入力して、実行させる命令・RUNを入力すると、パソコンは、まず行番号10を実行します。

行番号10は、READ　A、Bです。READ　A、Bは、READのあとにつづく変数AとBのデータを、DATAから読みとりなさいということですから、行番号80のDATAに示されている8と15というデータを、AとBにそれぞれ読みとります。

```
10 READ A,B

80 DATA 8,15
```

行番号10のREAD　A、Bを実行して行番号80のDATAから8、15のデータを読みとります

ＡとＢに読みとったデータ８と15を、つぎの行番号20のＡとＢに送り込みます。

行番号20は、Ａ＊Ｂ／２という計算式です。ここで８×15÷２が行われて、三角形の面積が求められます。求められた三角形の面積60は、左側のＣに入れられ、そして行番号30のＣに送り込まれます。

```
20 C=A*B/2
30 PRINT "ｻﾝｶｸｹｲﾉﾒﾝｾｷ";C
```

Ｃに入れられた60は、次に行番号30のＣに送り込まれます

ＰＲＩＮＴ　ＣでＣの中身60が表示されます

行番号30のPRINT　Ｃは、Ｃの中身を表示しなさいということですから、Ｃの中身の60が、ディスプレイの上に表示されることになります。

```
run
ｻﾝｶｸｹｲﾉﾒﾝｾｷ 60
```

行番号30を実行して、Ｃを表示するとパソコンは、次の行番号40の実行に移ります。行番号40はまた、READ　Ａ、Ｂです。これは、行番号10のREAD　Ａ、Ｂと同じです。しかし、行番号40のREAD　Ａ、Ｂは、行番号10のREAD　Ａ、Ｂと違って、行番号90の DATA から

15、20というデータを読みとります。これは、READで１度読みとられたDATAのデータは、２度使うことができないという仕組みによるものです。

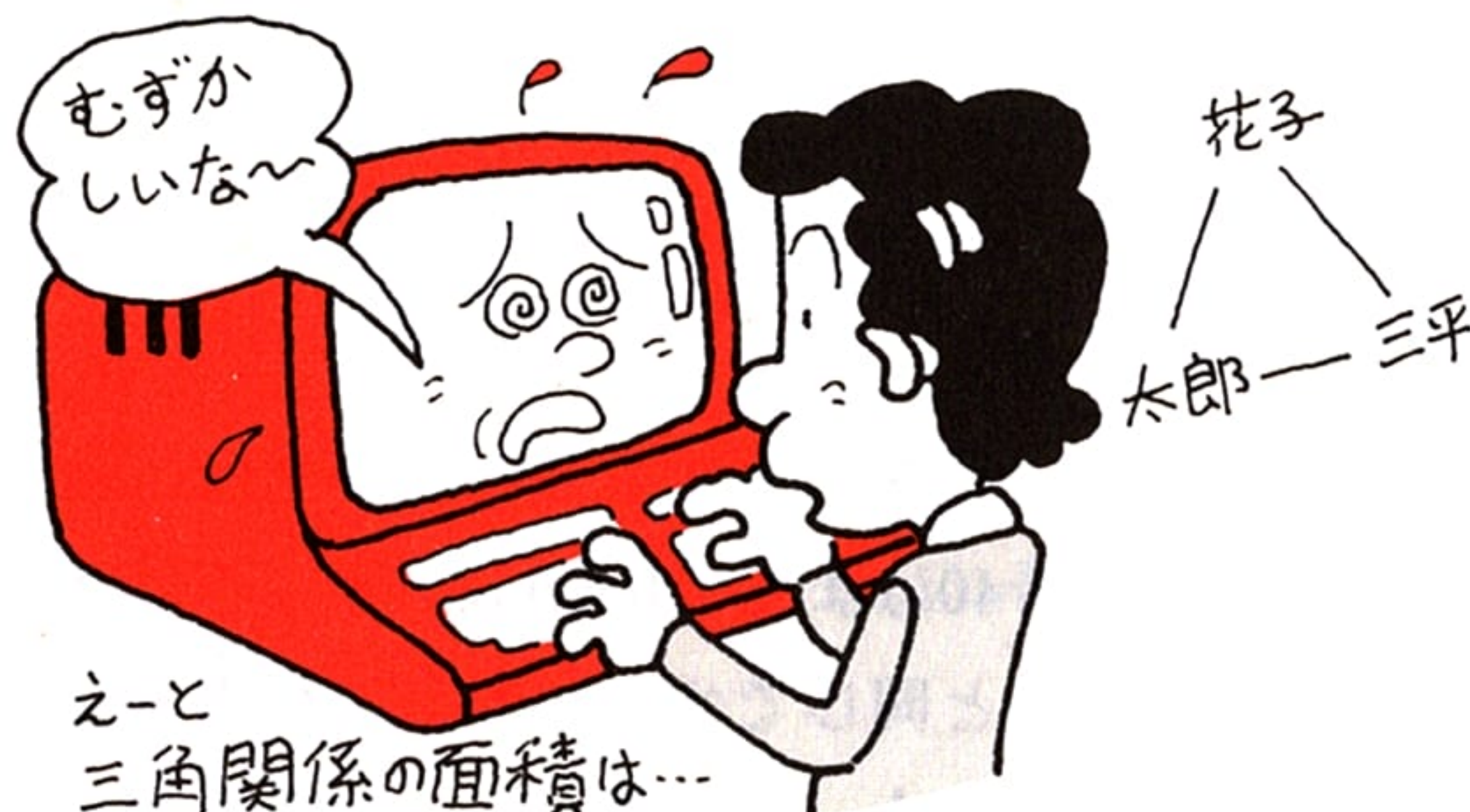
行番号40のＡとＢに読みとられたデータ15と20は、行番号50のＣ＝Ａ＊Ｂ／２のＡとＢに送り込まれ、15×20÷２が実行されます。計算された結果の三角形の面積150は、左側にあるＣに入れられ、それから、行番号60のPRINT　Ｃ　のＣに送り込まれます。PRINT　Ｃは、Ｃの中身を表示しなさいということですから、Ｃの中身150を表示して、おわります。

```
run
ｻﾝｶｸｹｲﾉﾒﾝｾｷ 150
```

さて、このプログラムでは、三角形の面積を２つ求めるということから、同じことが２度くり返されています。

行番号20と50のC＝A＊B／２

行番号30と60のPRINT　C

これをひとまとめにしてサブルーチンにし、GOSUB～RETURN

を使うと、つぎのプログラムのようになります。

GOSUB～RETURNを使ったプログラム

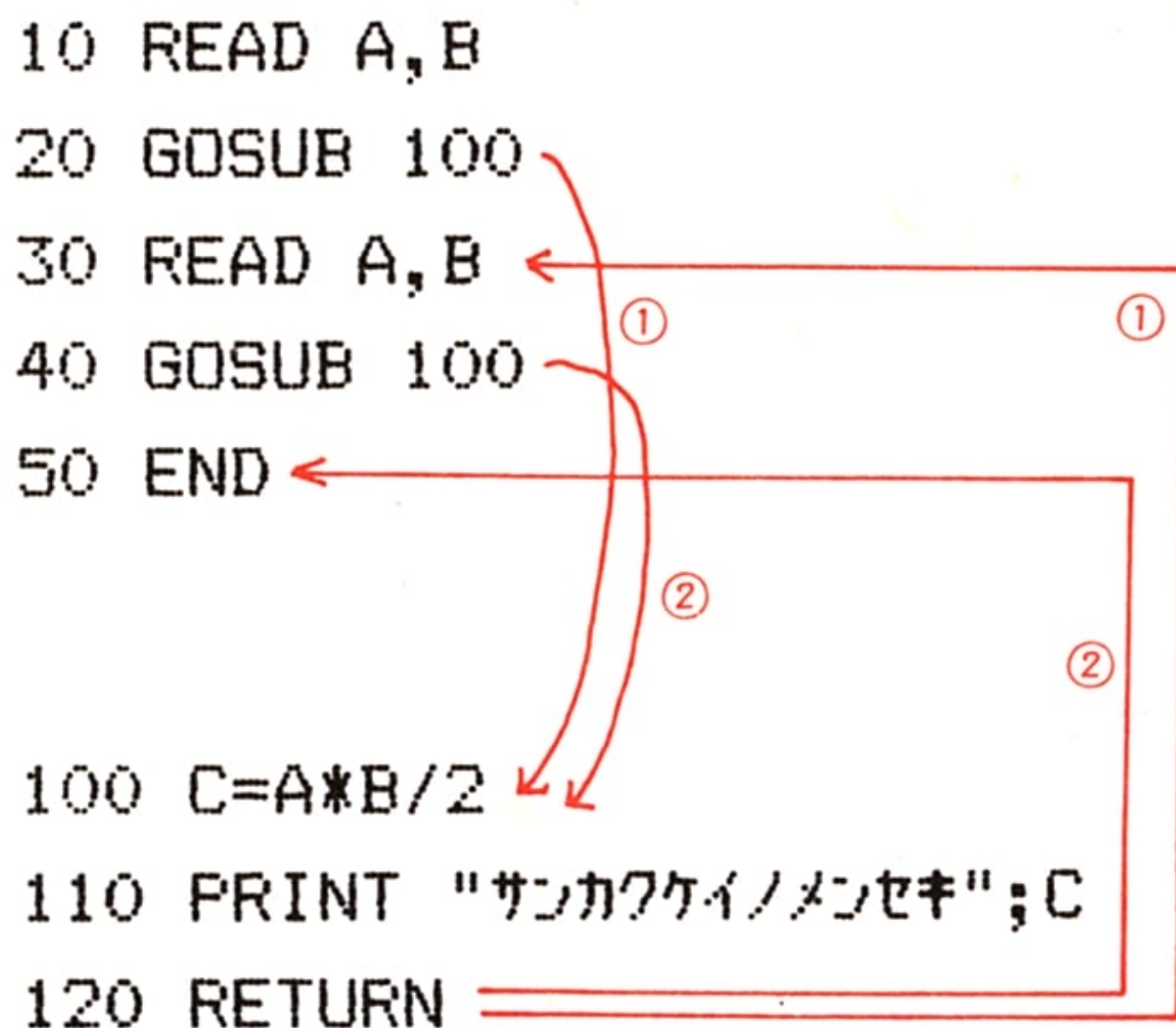
```
10 READ A,B
20 GOSUB 100
30 READ A,B
40 GOSUB 100
50 END
60 DATA 8,15
70 DATA 15,20
100 C=A*B/2
110 PRINT "ｻﾝｶｸｹｲﾉﾒﾝｾｷ";C
120 RETURN
```

さて、まえのプログラムで、2度くり返して使った〝C＝A＊B／2〞と〝PRINT　C〞をひとまとめにして、サブルーチンとしました。それが行番号100から120までです。

そして、メインルーチンで、このサブルーチンを必要とする行番号20と40に、GOSUB 100をおきます。RETURNは、メインルーチンへ戻れということですから、サブルーチンの最後の行番号120におきます。

では、このプログラムを入力して、実行させてみましょう。RUNを入力するとパソコンは、行番号10のREAD　A、Bを実行して、行番号60のDATAから8と15というデータを読みとり、つぎに行番号20の実行に移ります。行番号20は、GOSUB 100ですから、パソコンの実行は、サブルーチンの行番号100に飛びます。行番号100のC＝A＊B／2のAとBには、行番号10のAとBのデータ8と15が送り込まれます。そして8×15÷2が計算され、計算結果の60は、＝の左側のCに入れられます。つぎにそのCの中身60を、行番号110のCに送り

込みます。そして PRINT　C が実行されて、Cの中身60が表示されます。

パソコンは、Cの中身を表示しおわると、サブルーチンの最後の行番号 120 のRETURNを実行します。

RETURN は、メインルーチンに戻れです。RETURN のあとには、戻っていく先の行番号が示されていませんが、これは、パソコンの内部で、スタックポインタとスタックが働いて、戻るべき行番号を記憶しているようになっていますので、自動的に戻ります。戻る行番号は、GOSUB でサブルーチンへ飛んだ、つぎの行番号です。

この場合は、行番号20で飛びましたから、行番号30へ戻ります。

行番号30は、ふたたび、READ　A、Bです。パソコンは、これを実行して、こんどは行番号70のDATAに示されているデータ15と20を、AとBに読み込みます。

そして行番号40の GOSUB 100 で、サブルーチンに飛びます。行番号30で読みとられたAとBのデータ15と20は、行番号 100 のAとBに送り込まれて、A＊B／2 が実行されます。その結果の 150 は、＝の左側のCに入れられ、行番号 110 のCに送り込まれます。PRINT　C で、Cの中身 150 を表示すると、行番号 120 のRETURN です。この

場合は、行番号40でサブルーチンへ飛んだわけですから、つぎの行番号50へ戻ります。

行番号50の END で、パソコンはプログラムがおわりであることを知って、OK を表示します。

RETURN のあとに行番号を示さずに、自動的に戻るようにしてあるのは、このように戻るべき行番号が、常に異るからです。またこのおかげで、ひとつのものが何回もくり返し、使うことができるわけです。

カーソル制御のプログラム

```
10 PRINT CHR$(12)
20 FOR I=1 TO 17
30 PRINT TAB(I);I
40 NEXT I
50 FOR T=1 TO 1000
60 NEXT T
70 PRINT CHR$(12)
80 FOR I=17 TO 1 STEP -1
90 PRINT TAB(I);I
100 NEXT I
110 FOR T=1 TO 1000
120 NEXT T
130 PRINT CHR$(12)
```

GOTO

GOTOは、まっしぐらに飛んでいく

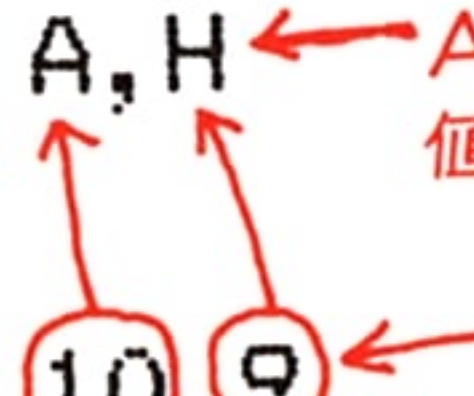
パソコンのプログラムの実行の仕方は、プログラムの各行の頭についている行番号の小さいものから大きいものへと、ひとつひとつ順番に実行していくというものです。

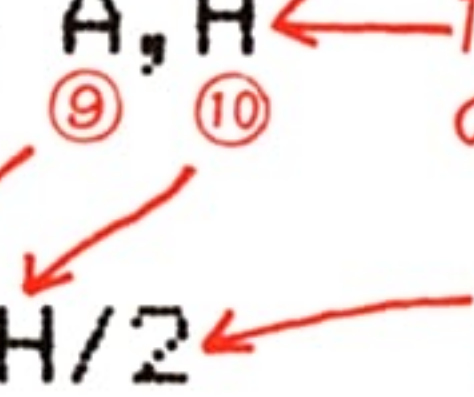
ところがＧＯＴＯを使うと、行番号の小さいものから大きいものへという、プログラムの実行の仕方をくつがえしてしまって、すでに実行してしまった行番号へ戻して、ふたたび実行させたり、また中間にある行番号を無視して、はるか遠くにある行番号へ飛ばして、実行させたり、いろいろにプログラムの実行の仕方を変えることができます。

では、実際にプログラムでみてみましょう。つぎのプログラムは、三角形の面積を求める簡単なプログラムです。

```
10 READ A,H   ← 変数AとHの値をDATAから読みとりなさい
20 S=A*H/2    ← Aの値とHの値をかけて2で割り、
                その結を左側のSに入れなさい
30 PRINT S    ← Sの値を表示しなさい
40 GOTO 10    ← 行番号10に飛びなさい
50 DATA 10,9  ← 1回目にREAD　A、Hに読み込まれます
60 DATA 8,6   ← 2回目にREAD　A、Hに読み込まれます
70 DATA 9,12  ← 3回目にREAD　A、Hに読み込まれます
80 END
```

このプログラムでは、行番号40にＧＯＴＯが使われています。

```
40 GOTO 10
```

へ飛べ　　飛んでいく先の行番号

ＧＯＴＯとは、～へ飛べといった意味で、ＧＯＴＯのあとには必ず、飛んでいく先の行番号がつづきます。これがＧＯＴＯの形です。

このプログラムの場合は、40　ＧＯＴＯ　10ですから、行番号40から、行番号10へ飛んでいきます。

さて、このプログラムを入力して、ＲＵＮを入力すると、一番最初の行番号10　ＲＥＡＤ　Ａ、Ｈを実行します。

ＲＥＡＤ　Ａ、Ｈの意味は、ＲＥＡＤのあとにつづくＡとＨに、ＤＡＴＡに示されている値を読みとりなさい、ということです。そこでまず、３つあるＤＡＴＡのなかの、１番若い行番号のＤＡＴＡから値が読みとられます。

行番号10のＡとＨに読みとられた10と９という値は、つぎに、行番号20のＡとＨに送り込まれます。

行番号20のＡとＨに10と９の値が送り込まれると、計算式にもとづいて、計算が行なわれます。Ａ＊Ｈ／２は、Ａ×Ｈ÷２のことで、パソコンでは、×は＊、÷は／といった記号が使われます。

10×９÷２＝45の45は、いったん＝の左側にあるＳに入れられます。そして、つぎにこのＳの中身45は、行番号30のＰＲＩＮＴ　ＳのＳに送り込まれます。

行番号30のＰＲＩＮＴ　Ｓは、Ｓの中身を表示しなさいということですから、Ｓの中身45が、ディスプレイに表示されます。

さて、行番号40のＧＯＴＯ　10です。

パソコンは、ＰＲＩＮＴ　Ｓで、45を表示すると、つぎに行番号40の実行に移ります。

行番号40は、ＧＯＴＯ　10、つまり行番号10へ飛べということですから、行番号10へ実行が移って、ふたたびＲＥＡＤ　Ａ、Ｈを実行します。

今度は、行番号60のＤＡＴＡから、８と６といった値を読みとって、まえと同じように、８×６÷２を実行します。そして計算結果の24を表示すると、また40　ＧＯＴＯ　10を実行して、行番号10へ飛びます。

つぎは、行番号70のＤＡＴＡから、９と12の値を読み込み、９×12÷２を実行し、その結果の54を表示します。

このプログラムでは、ＤＡＴＡは３つしかありません。したがって、パソコンの実行は、行番号80のＥＮＤに移るのかというと、そうではありません。なぜなら、行番号40のＧＯＴＯ　10で、また行番号10に実行が移ってしまうからです。

行番号10に飛んだ以上、もちろんＲＥＡＤ　Ａ、Ｈを実行します。ところが、もう読みとるＤＡＴＡはありません。１度、使ってしまったＤＡＴＡの値は、２度使うことができないことになっています。

その結果、パソコンは、ピーッという音を発して、

```
Out of DATA in 10
```

というメッセージを表示します。そして実行を終了するのです。

Out of DATA in 10という意味は、行番号10のＲＥＡＤ　Ａ、Ｈに読みとる値が、ＤＡＴＡに用意されていないということです。

さて、つぎに示すのが、実行結果です。

```
run
 45 ←—— １回目の実行結果
 24 ←—— ２回目の実行結果
 54 ←—— ３回目の実行結果
Out of DATA in 10
```

これまでおはなししたように、ＧＯＴＯが実行されると、ＧＯＴＯのあとに示された行番号に無条件に飛んでいってしまい、飛んでいった先の行番号のステートメントを実行します。

ＧＯＴＯのこのような性格から、ＧＯＴＯのことを無条件飛び越しとか、無条件ジャンプとか呼んでいます。

INSキーとDELキー

INSキーとDELキーは、プログラムの修正に活躍する

キーボードの上にあるINSキーとDELキーは、入力を忘れた文字などを挿入したり、余分な文字などを削除するときに使います。

INSキーのINSは、INSERT（インサート）の略で、挿入するという意味です。DELキーのDELは、DELETE（デリート）の略で、削除するという意味です。

まず、INSキーです。INSキーは、さきに説明したように、抜けてしまった文字などを追加するときに使います。

```
10 INPUT A
20 PRNT A
30 END
```

Iが抜けている

このプログラムを実行させると、エラーになります。なぜなら行番号20のPRNTは、PRINTでなくてはならないからです。つまりIが抜けてしまっているわけです。そこで、RとNの間にIを挿入することにします。

Iを挿入するには、カーソルを矢印のキーで動かして、Nの上にもってきます。

```
20 PRNT A
```

カーソルをNの上にもってくる

カーソルをNの上にもってきたら、INSキーを叩きます。するとカーソルが半分ぐらいの大きさ（▬）に縮小します。

```
20  PRNT  A
```

カーソルが半分の大きさになる

カーソルが半分の大きさに縮小されたら、文字などを挿入してもよいということですから、Ｉのキーを叩きます。すると、

```
20  PRINT  A
```

Ｉのキーを叩くと、Ｉが挿入される

Ｉが挿入されて、Ｎ以降の文字はひとつ右にずれます。Ｎの上のカーソルは、まだ半分の大きさに縮小したままです。挿入が終わったらRETURNキーを叩きます。するとカーソルはもとの大きさに戻って、つぎの行の先頭の位置に移動します。

この場合はＩだけの挿入でしたから、Ｉを挿入してRETURN キーを叩きましたが、挿入する文字は１文字だけではなく、何文字でも必要なだけ挿入することができます。

```
10 PRINT "ﾜﾀｼﾊ ﾊﾅｺﾃﾞｽ｡"
20 END
```

タチバナを挿入する

行番号10の引用符「″」で囲まれているなかのハナコのまえに、タチバナを挿入することにします。カーソルを矢印のキーで移動して、ハの上にもってきます。

```
10  PRINT  "ワタシハ  ハナコデス。"
```

カーソルをハの上にもってくる

カーソルをハの上にもってきたら、INSキーを叩きます。するとカーソルは、半分の大きさになりますから、

```
10  PRINT  "ワタシハ  ハナコデス。"
```

カーソルが半分の大きさになる

キーを叩いてタチバナと入力します。タチバナと入力すると、ハ以降が右へ５文字分移動して、その間にタチバナが挿入されます。

```
10  PRINT  "ワタシハ  タチバナハナコデス。"
```

タチバナが挿入される

INSキーとDELキー

タチバナを挿入したら、RETURNキーをたたきます。するとカーソルはもとの大きさに戻って、つぎの行の先頭の位置に移動します。

INSキーを押すと、カーソルが半分の大きさになると説明しましたが、どの機種もカーソルが半分の大きさになるとはかぎりません。機種によっては、そうならない機種もあるようですから、その機種のマニュアルを見て確めてください。

さて、つぎはDELキーです。DELキーは、さきに説明したように、不必要な文字などの削除に使います。

```
10 INNPUT A
20 PRINT A
30 END
```

Nがひとつ余計なので削除する

このプログラムを実行させると、これもエラーになります。なぜなら行番号10のINPUTが、INNPUTになっているからです。つまり、Nがひとつ余計です。そこで、余計なNを削除します。

まず、矢印のキーでカーソルを動かして、削除するNの上にもってきます。

```
10  INNPUT  A
```

カーソルを削除するNの上にもってくる

この場合はおなじNですから、どちらを削除してもよいので、最初のNの上にカーソルをもってきています。カーソルを削除するNの上にもってきたら、DELキーを叩きます。すると、カーソルが置かれた位置のNが削除されて、つぎのN以降が1文字分左へ移動します。そして、カーソルはつぎのNの上に移動します。

```
10  INPUT  A
```

DELキーを叩くと、カーソルの位置のNが削除される

削除が終了したら、RETURNキーを叩きます。

CONT

CONTは、ふたたびプログラムの実行を開始させる

プログラムの実行を途中で止めるには、プログラムのなかにSTOPということばを書いておいたり、キーのうえにあるSTOPキーとSHIFTキーを一緒に押します。

プログラムを実行しているときにSTOPに出合うと、パソコンは、

```
Break in 50
```

行番号50で、プログラムの実行が打ち切られたというメッセージ

といったように、プログラムの実行が中断された行番号を表示して、実行を中断します。STOPでは、パソコンはファイルを閉じていません。ただ、一時プログラムの実行を中断して、つぎの命令を待つ、という状態になっているだけです。

したがって、プログラムのデータや計算過程などのチェックをすることができます。

このSTOPを使って中断させたプログラムの実行を、ふたたびつづけさせる命令がCONTです。CONTはContinue（コンテニュー）の略で、中断後ふたたび続けるという意味です。

[C][O][N][T]とキーボードのキーを叩いて入力し、RETURNキーを押せばよいのです。またSHIFTキーとファンクションキーのF·3キーを一緒に押しても同じです。ふつう[f·3]キーの裏にある[f·8]キーがCONTの働きをするキーです。

「,」コンマ

コンマで区切ると、間隔をあけて表示する

日常の生活で私たちが、さして気にもとめないで使っているコンマ「,」が、パソコンでは、あまりにも重要な働きをするので、びっくりするはずです。

まずその例を、お目にかけましょう。

```
10 A=-10
20 B=5
30 PRINT A,B
40 END
run
-10            5
```


Aの値は－10で、Bの値は5です。この値は行番号30のAとBに送られて、ディスプレイに、AとBの値を表示しなさいというPRINT A, Bが実行された結果が、runの下に示されているものです。－10の－から5までの間隔は、14字分あります。

ここで、このコンマは、どういう働きをしたのでしょうか。

実は、このコンマは、Aの値とBの値を、一定間隔をとって表示しなさいという、指示をしたのです。コンマのこの働きで、うえの例に示すように、文字が一定の間隔をとって表示されたわけです。

では、行番号30のAとBの間にコンマを2つ入れて

```
30 PRINT A, ,B
```

としたらどうなるでしょうか。この場合は、AとBの間隔が14字分の倍とられて、2行に並んで表示されます。

```
run
-10
 5
```

コンマのこの働きは、文字を表示させるときも、おなじです。

文字を表示させるには、「〃」クォーテーションマークで、文字を囲まなければなりません。こうすれば文字は、そのとおりに表示されます。記号のときも、おなじです。これを、ストリングといいます。ストリングを入れる変数は、AやBのあとに「$」ドルマークをつけます。

```
10 A$="ﾀﾛｳ"        タロウをA$に入れなさい
20 B$="ﾊﾅｺ"        ハナコをB$に入れなさい
30 PRINT A$,B$     A$のタロウとB$のハナコを表示しなさい
40 END
run
ﾀﾛｳ           ﾊﾅｺ
```

文字のときは－記号はつかないので、一字あかない

14字

このようにコンマは、数値や文字を一定の間隔をあけて表示するように働きます。これとは逆に、文字や数値をつづけて表示させる働きをするのが、「；」セミコロンです。

CHR$(12)

CHR$(12)は、画面の表示をすべて消す

ゲームのプログラムのなかでよくみかけるものに、CHR$(12) というものがあります。CHR$(12)は、PRINTといっしょに使われます。PRINT　CHR$(12)というかたちです。

このPRINT　CHR$(12) をパソコンが実行すると、それまでディスプレイの画面に表示されていたものが、すべて消えてしまいます。

つぎのプログラムは、文字をある一定の時間表示して消し、つぎの文字が表示されるというプログラムです。

```
10 PRINT "ｷｮｳﾉﾔｷｭｳ"
20 FOR I=1 TO 1000
30 NEXT I
40 PRINT CHR$(12)
50 PRINT "ｷｮｼﾞﾝ ﾀｲ ﾔｸﾙﾄ"
60 FOR I=1 TO 1000
70 NEXT I
80 PRINT CHR$(12)
90 PRINT "ｷｮｼﾞﾝ ﾀｲｻﾃﾞｶﾂ"
100 FOR I=1 TO 1000
110 NEXT I
120 PRINT CHR$(12)
130 PRINT "ｵﾜﾘ"
```

クォーテーションマークで囲まれた文字は、そのとおりに表示されます

Iが1から1000になるまでくり返し処理しなさい。処理している時間、キョウノヤキュウが表示されています

画面表示を消します

前頁のプログラムを入力して、RUNさせると、パソコンはまず行番号10を実行して、キョウノヤキュウをディスプレイの画面に表示します。「〃」クォーテーションマークで囲まれた文字は、ストリングといって、そのとおりに文字が、画面に表示されます。

パソコンは、画面に文字を表示すると、つぎに行番号20と30のFOR～NEXTの実行に移ります。このFOR～NEXTは、キョウノヤキュウという文字を、Iが1から1000になるまでの時間、画面に表示させておくためのものです。

パソコンは、Iが1000になるまで、FOR～NEXTをくり返すと、行番号40のPRINT　CHR$(12)の実行に移ります。

PRINT　CHR$(12)は、それまで画面に表示されているものを、すべて消すといった働きをするものですから、行番号40が実行されると、キョウノヤキュウという文字が消えます。

つぎにパソコンは、行番号50のPRINT〝キョジン　タイ　ヤクルト〃を実行して、その文字を表示します。

おなじ方法で、その文字を消します。そして、キョジン　タイサデカツを表示し、その文字を消して、オワリを表示します。

このPRINT　CHR$(12)は、29頁で説明しているCLSとおなじ働きをします。

CHR$(X)

CHR$(X)は、キャラクタ・コードを文字へ変換する

A、B、Cなどのアルファベット、1、2、3などの数字、ア、イ、ウなどのカタカナ、+、−、=などの記号を総称して、キャラクタと呼びます。このキャラクタは、すべてキャラクタ・コードをもっています。

これらキャラクタを、キャラクタ・コードに変換するときに使うのが、12頁でおはなししたASCです。

CHR$は、ASCとは逆に、キャラクタ・コードをキャラクタ、つまり文字や記号などに、変換するときに使うものです。

たとえばキャラクタ・コード「89」の文字が、なんであるか知りたいときは、CHR$を使って、つぎのようにすればよいのです。

CHR$のかたちは、CHR$(X)です。カッコのなかのXに入るものは、0から255までの整数となっていますが、実際には、0から255までは使われていません。パソコンで実際に使われているコードは、1から253までです。

また、CHR＄（X）のXに、直接、数値を入れずに、つぎのような方法でも、キャラクタを求めることができます。

```
10 A=25
20 B=41
30 PRINT CHR$(A+B)
40 END
run
B
```

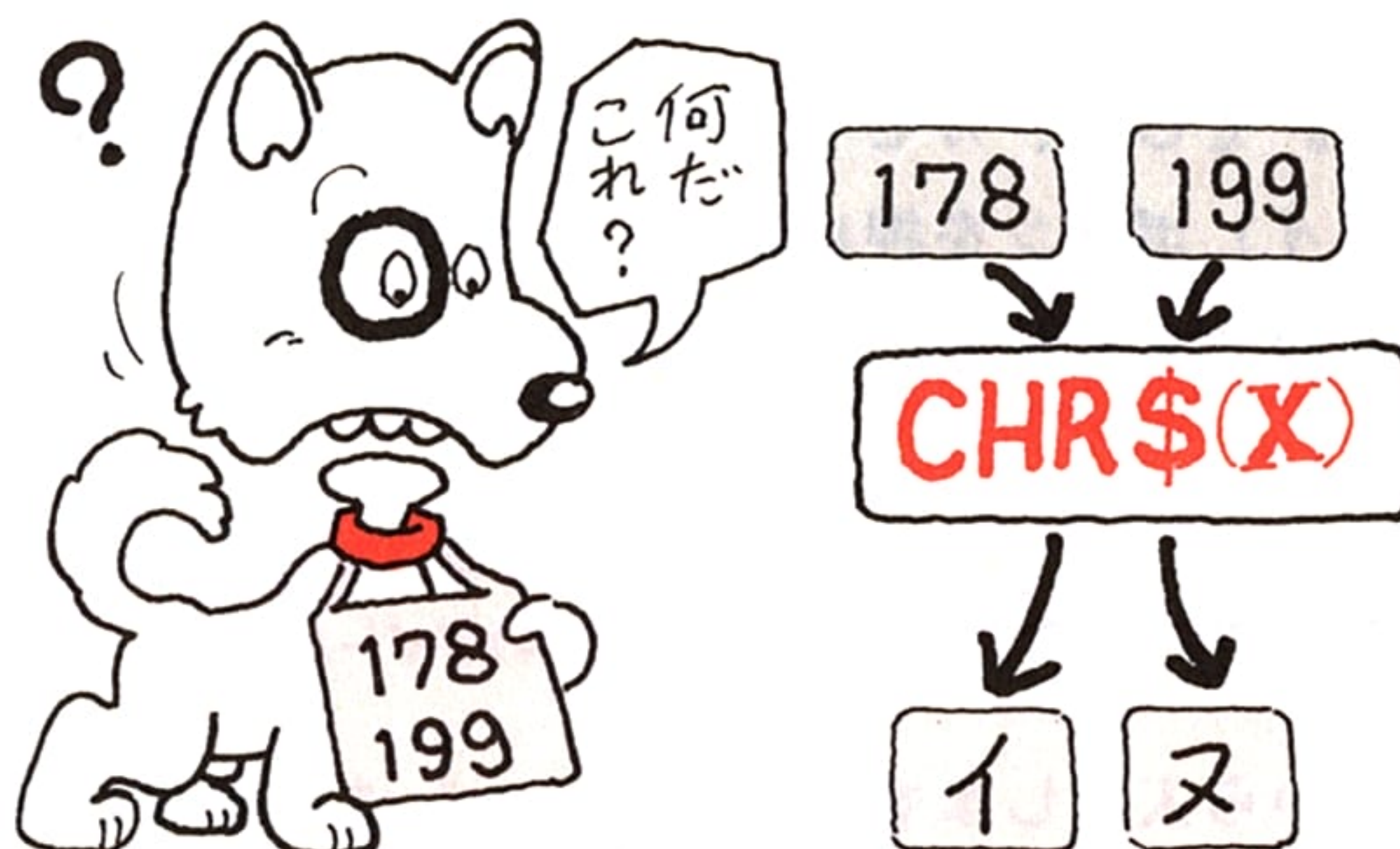
Aの値25とBの値41が足し算されてその結果の66を、CHR＄でキャラクタに変換します

66のキャラクタ

```
10 A=22
20 PRINT CHR$(A*A*.5)
30 END
```

Aの値22が、それぞれAに入り、22×22× 0.5が行なわれて、その結果の 242が、CHR＄でキャラクタに変換されます。

CHR＄のかたち

CHR＄（X）　Xには、変数、数値定数、計算式が入ります。いずれの場合も整数です。整数でない場合は小数点以下が切り捨てられます。

CSAVE

CSAVEは、カセットテープへプログラムを記録する

キーボードのキーをひとつひとつ叩いて、パソコンにプログラムを入力するのは、たいへんにくたびれるものです。

せっかく苦労して入力し、間違ったところなどを訂正して完成したプログラムでも、パソコンの電源を切ったとたん、メモリから消滅してしまいます。

そこで、プログラムを保管するために使われるのが、カセットテープレコーダです。実際に、カセットテープレコーダを使ってみると、とても便利なものです。

ところでCSAVEは、パソコンのメモリのなかに記憶されているプログラムを、カセットテープに記録させるときに使うコマンド（命令）です。

カセットテープにプログラムを記録させるには、まず、キーボードのキーを叩いて、

```
MOTOR 0
```

と入力します。そして、カセットテープレコーダのREC（レコード）とPLAY（プレイ）ボタンを押します。こうすると、カセットテープレコーダは、記録できる状態になりますが、テープはまだまわりません。つぎに

とキーボードから入力します。〝――――〟この部分には、プログ

ラムのタイトル（ファイル名）を入れます。このタイトルは、何文字でもつけることができますが、パソコンは頭から6文字だけでタイトル名を判断しますから、簡単で、しかもプログラムの内容を表しているようにつけなければなりません。タイトル名は、英字、カナ、数字、記号などでつけることができますが、英字の場合、大文字、小文字を区別してつけます。パソコンは、〝OSERO〟、〝osero〟を別のプログラムと判断してしまうからです。また、このタイトルは、カセットテープに記録されているプログラムを、パソコンのメモリへ戻すときにも使いますから、おぼえやすいものにすることです。

たとえば、数あてゲームなら、つぎのようになります。数あては、カス゛アテとなって、すでに5文字になりますから、ゲームまでは入りません。そこで、ゲームの頭文字をとって、Gとします。

```
csave "カズ゛アテG"
```


このようにキーボードから入力したら、RETURNキーを押します。すると、テープが回り、カセットテープへの記録が開始されます。

カセットテープへの記録がおわると、自動的にストップして、ディスプレイの画面のうえに、OKが表示されます。

OKが表示されたら、はじめにMOTOR　0と入力したのに対して、こんどは

MOTOR 1

と入力します。そうすると、はじめてカセットテープの巻き戻しなどが可能になります。

プログラムをカセットテープに記録するときに、注意することは、テープレコーダのカウンタの目盛をみて、何番から何番までにプログラムを記録したか、ノートなどにメモしておくことです。こうしておくと、使われているカセットテープの部分がわかり、同じところにまたプログラムを記録するなどといったことを、防ぐことができます。

CLOAD?

CLOAD ?は、メモリの内容とテープの内容を比較させる

パソコンのメモリからカセットテープへ、プログラムを記録しおわったら、そのプログラムが、正確に記録されたかどうか、確かめることが大切です。

なぜなら、記録しているときにノイズ（雑音）が入ってしまって、正確に記録されていなかったり、また、テープにキズがあって、正確に記録されなかったといったことが、ままあるからです。

パソコンのメモリからプログラムを、カセットテープに記録させても、メモリのなかのプログラムは消滅しませんから、それとカセットテープへ記録したプログラムとが一致しているかどうか、パソコンに比較させて、確かめるわけです。これをベリファイといいます。

CLOAD ?は、このベリファイを行うときに使うコマンド（命令）です。

CLOAD ? のあとにつづく〝——〟この部分には、ベリファイを行うプログラムのタイトル（ファイル名）が入ります。

たとえば、つぎのようになります。

```
cload? "カズアテG"
```

このように、コマンドとタイトルをキーボードから入力したら、つぎに、カセットテープレコーダのPLAYボタンを、押します。

するとパソコンは、まず〝——〟この部分に指定されたタイトル

名のプログラムを、カセットテープのなかから探します。タイトルを探しだすと、ディスプレイの画面のうえに、

Found：カズ　アテG

と表示します。FOUND（ファウンド）とは、発見という意味です。

そして、ベリファイをはじめます。

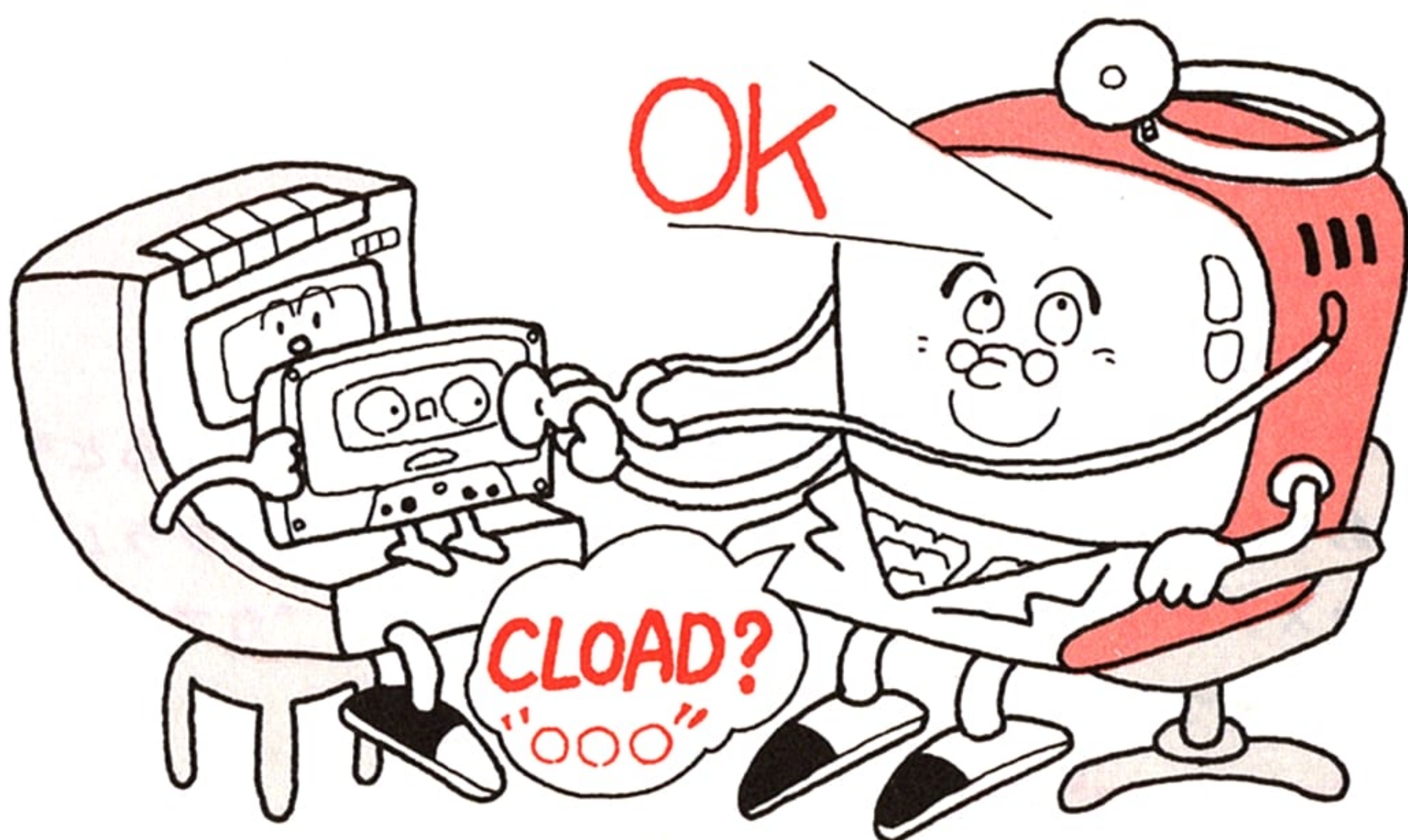
ベリファイを行って、メモリのなかのプログラムと、カセットテープに記録したプログラムとが一致していると、ディスプレイの画面のうえに、

OK

が表示されます。もし一致していないときは、

Verify　error

が表示されます。

Verify errorが表示されたときは、カセットテープへの記録が失敗しているわけですから、もう一度、カセットテープへの記録を行わなければなりません。

CLOAD

CLOADは、テープのプログラムをパソコンのメモリへ戻す

CSAVEを使って、パソコンのメモリから、カセットテープに記録したプログラムを、ふたたびパソコンのメモリのなかへ戻すときに使うコマンド（命令）が、CLOADです。

カセットテープからメモリのなかへプログラムを戻すときは、まず、キーボードから

```
MOTOR 0
```

と入力します。そして、カセットテープレコーダのPLAYボタンを押します。この状態では、まだテープはとまったままです。

つぎに、キーボードから

```
cload "____"
```

カセットテープからメモリのなかへ戻すプログラムのタイトルを入れます

と入力します。〝——〞の部分には、パソコンのメモリに戻すプログラムのタイトル（ファイル名）を入れます。

たとえば、つぎのようにです。

```
cload "ｶｽﾞｱﾃG"
```

このように入力したら、RETURN キーを押します。するとテープは、まわりはじめます。そしてパソコンは、目的のプログラムのタイトル〝カズアテG〞をテープのなかから探します。〝カズアテG〞のタイトルがみつかると、

```
Found ｶｽﾞｱﾃG
```

とディスプレイの画面のうえに表示されます。そして、LOAD（ロ

ード）が開始されます。

ロードがおわると、

OK

がディスプレイの画面のうえに表示されます。

このOKが表示されれば、カセットテープのプログラムをパソコンのメモリに戻すロードに成功したわけですが、パソコンは、ピーッと音を発して、つぎのようなメッセージをディスプレイの画面に表示することもあります。

Device I/O error

このメッセージは、ロード中にエラーが発生して、ロードが失敗したという意味のものです。

このようなメッセージが表示されたときは、もう1度ロードをやり直す必要があります。

パソコンのメモリから、カセットテープへプログラムを記録しおわったあと、正確に記録されたかどうかを確かめるための、ベリファイを行っていれば、ロードの失敗は、単純なものなので、もう1度やり直す程度で、解決することができるはずです。

四則演算

四則演算は、足し算、引き算、かけ算、割り算のこと

四則演算とは、足し算、引き算、かけ算、割り算のことです。
この四則演算に使う記号を、四則演算記号といいます。
四則演算記号は、数学とBASICとでは、多少異なります。

数学とBASICの四則演算記号

	足　す	引　く	かける	割　る
数　学	＋	－	×	÷
BASIC	＋	－	＊	／

数学とBASICで、記号が異なるところは、うえの表をみてもわかるように、数学では、かける、割るに、×、÷という記号を使いますが、BASICでは、かける、割るに、＊、／を使うことです。

これらの四則演算記号を使って、BASICで数式を書いてみると、つぎのようになります。

A＋B　（足し算）
A－B　（引き算）
A＊B　（かけ算）
A／B　（割り算）

うえの式のように、四則演算記号がひとつしかないときは、問題はありませんが、つぎのように四則演算記号が、いくつも使われているときには、パソコンは、決められた順番で計算を行います。

```
A*B+C/D
A*B-C/D
```

＊と／が先に計算されます。
＋と－は、そのつぎに計算されます

このような式のときは、A＊BとC／Dが、B＋CやB－Cよりも先に計算されます。

四則演算記号の計算順序は、＊と／が、＋や－よりも先に計算されるということです。＊と／の順序はおなじで、どちらが先に計算されるかは、決まっていません。おなじ数式のなかにあるときは、左から右へという順序で、計算が行われます。＋と－の場合もおなじです。

ただし、数式のなかに、カッコでくくられたところがあると、カッコのなかから計算が行われます。たとえカッコのなかの四則演算記号が、＋や－であってもです。

```
(A-B)/C
(A+B)*C
```

カッコのなかから計算が行われます

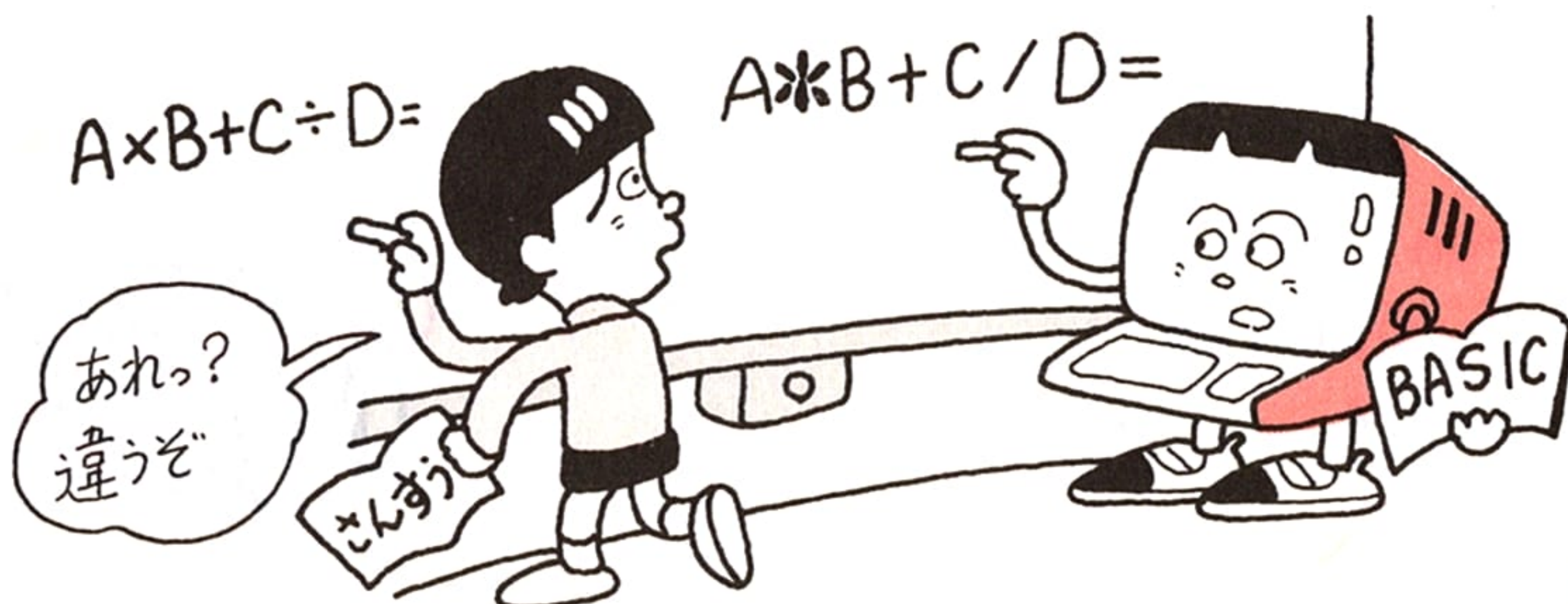
うえの例の場合では、カッコ内の－、＋が計算されてから、／や＊の計算が行われます。

このように、四則演算記号に計算していく順序があるので注意して数式を作らなければなりません。たとえば、AからBを引いた結果に、Cをかける場合、

```
A-B*C
```

とすると、間違いです。このようなときは、

```
(A-B)*C
```

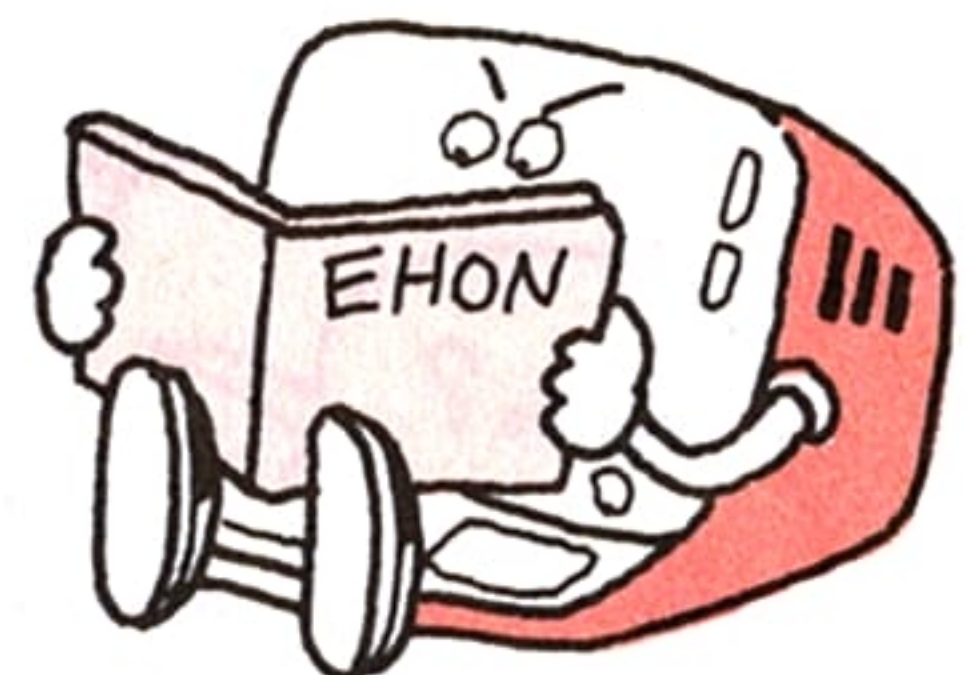
としなければなりません。

では、A－B＊Cと（A－B）＊Cとでは、どのように計算結果が違ってくるか、つぎのプログラムを実行させてみてみましょう。

```
10 INPUT A,B,C
20 S=A-B*C
30 X=(A-B)*C
40 PRINT S,X
50 END
run
? 20,10,30
-280            300
```

A－B＊Cとしたときの結果は、－280ですが、（A－B）＊Cとしたときの結果は、300です。

このように、数式の作り方が間違っていると、求める結果が得られないといったことになります。

もうひとつ例をあげておきましょう。

```
10 INPUT A,B,C
20 S=A+B/C
30 X=(A+B)/C
40 PRINT S,X
50 END
run
? 20,10,25
 20.4           1.2
```

ストリング

ストリングは、文字をクォーテーションマークで囲む

「私は山田太郎です」といった文字を、パソコンのメモリのなかに記憶させ、ディスプレイの画面のうえに表示させるには、その文を、「〝」クォーテーションマークで囲みます。

〝ワタシハ　ヤマダ　タロウデス〞

このように、クォーテーションマークで囲んだ文字の集まり（文字列）を、ストリングといいます。

```
10 PRINT "ﾜﾀｼﾊ ﾔﾏﾀﾞﾀﾛｳ ﾃﾞｽ"
20 END

run
ﾜﾀｼﾊ ﾔﾏﾀﾞﾀﾛｳ ﾃﾞｽ
```

もちろん、このように意味のある文字の集まり（文字列）だけを、ストリングというわけではありません。つぎのように、

〝AEY×BDO〞

〝＋／－＊％＆；〞

〝５６？！＃７８〞

であっても、ストリングです。

つまり、パソコンにとっては、文字の意味とか、文字の種類などは問題ではなく、クォーテーションマークで囲まれていれば、ストリングとして扱い、ストリングとしてメモリのなかに記憶して、表示するわけです。

ストリング変数

ストリング変数は、文字列(ストリング)を扱う

パソコンでは、数値を処理するのに、変数を用います。

うえのプログラムのA、B、Cは、通常の変数です。正確には数値を扱う変数なので、数値変数といいます。

このように、数値を扱うものに、通常の変数があるのとおなじように、ストリングを扱うものに、ストリング変数があります。ストリング変数は、文字変数ともいいます。

ストリングとは、まえの頁でおはなししたように、

〝ワタシハ　フシ゛サクラテ゛ス″

「〝」クォーテーションマークで囲まれた、文字の集まり（文字列）のことです。

ストリングを扱うストリング変数のかたちは、A、B、Cといったアルファベットのあとに、「$」ドルマークがついたものです。

A＄、B＄、C＄

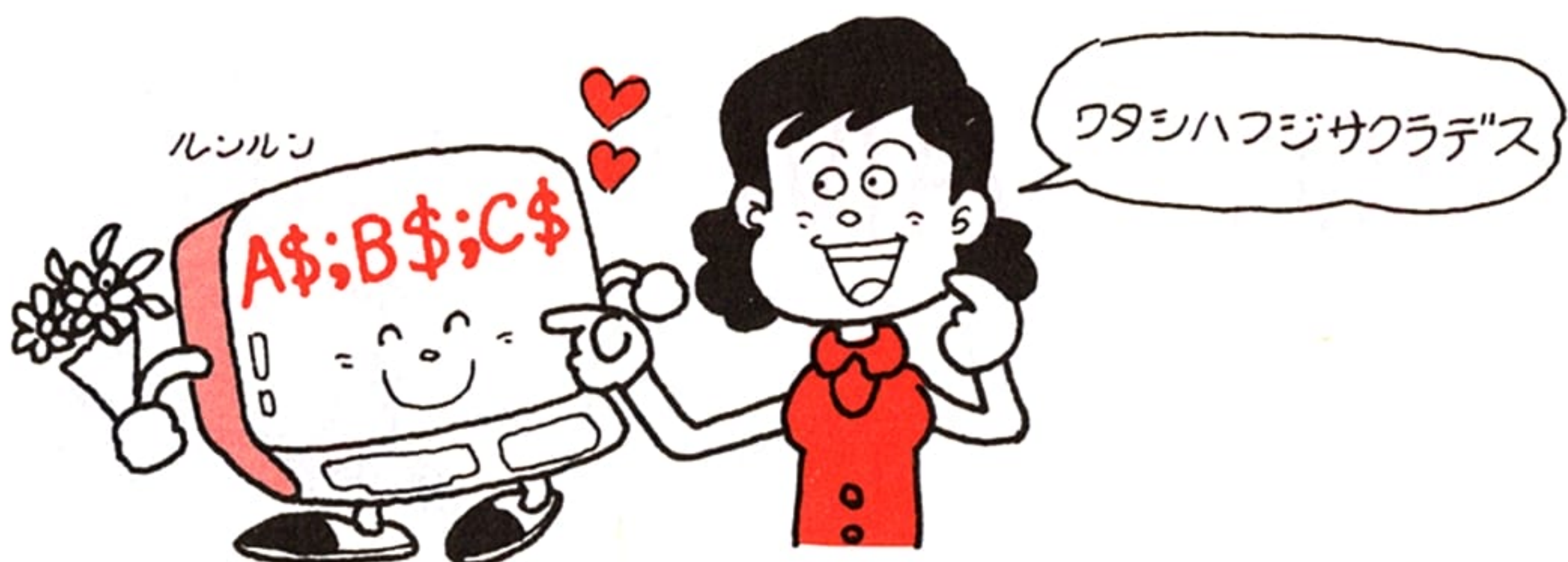
また、アルファベットに数字がひとつついたものに、＄マークがつく場合もあります。これも、ストリング変数です。

A１＄、A２＄、B１＄、B２＄

ストリング変数の働きは、通常の変数の働きとおなじです。

```
10 A$="ﾜﾀｼﾊ"     ←ワタシハをA＄に入れなさい
20 B$="ﾌｼﾞ"      ←フジをB＄に入れなさい
30 C$="ｻｸﾗﾃﾞｽ"   ←サクラデスをC＄に入れなさい
40 PRINT A$;B$;C$ ←A＄、B＄、C＄の中身を表示しなさい
50 END              (;)セミコロンをつけると、文字をつづけて表示します
run
ﾜﾀｼﾊﾌｼﾞｻｸﾗﾃﾞｽ      ←実行結果です
```

行番号10のA＄には、ワタシハが入ります。行番号20のB＄には、フジが入ります。そして、行番号30のC＄には、サクラデスが入ります。PRINTは、ディスプレイに表示しなさいということですから、行番号40が実行されると、A＄、B＄、C＄の中身がディスプレイに表示されることになります。実行結果は、run のつぎに示されているとおりです。

このように文字がつづけて表示されたのは、A＄；B＄；C＄の間

に、セミコロンが使われているからです。

```
10 A1$="BOY"
20 A2$="AND"
30 A3$="GIRL"
40 PRINT A1$;A2$;A3$
50 END
 run
 BOYANDGIRL
```

うえの例は、ストリング変数をＡ１＄、Ａ２＄、Ａ３＄にし、ストリングに英文を使ったものです。ストリングに英文を使うときは、カタカナと違って、注意することがあります。それは、英文には、大文字と小文字があるということです。ふつうは、小文字で入力して、LISTをとると大文字に変換されています。しかし、小文字をクォーテーションマークで囲むと、小文字のままで、大文字に変換されませんから、大文字で表示したいときは、入力するときにSHIFTキーを押して、大文字で入力しておかなければなりません。

ところで、うえのプログラムの実行結果をみてください。

```
run
BOYANDGIRL
```

なんとなく、みにくくはないでしょうか。もし、つぎのようになっ

ていると、みやすいはずです。

```
BOY AND GIRL
```

このようにするには、クォーテーションマークで囲んだなかに、1字分空白をとればよいのです。

```
10 A1$="BOY□"
20 A2$="AND□"
30 A3$="GIRL"
40 PRINT A1$;A2$;A3$
50 END
run
BOY AND GIRL
```

一字分空白をとります

空白は、文字とおなじように扱われます

クォーテーションマークで囲んだなかの空白は、文字とおなじに扱われます。ですから、クォーテーションマークで囲んだなかなら、どこでも、また、すきなだけ空白をつくることができます。

これまでは、文字を例にしてみてきましたが、つぎに数字を例にしてみましょう。

```
10 A$="238"
20 B$="569"
30 C$=A$+B$
40 PRINT C$
50 END
run
238569
```

このように、クォーテーションマークで囲むと、数字は、数値としての意味がなくなってしまうので、足し算をしても、ただ数字が並ぶだけです。98頁の例は、数値変数に数値としての数字を入れたものです。くらべてみると、その相違がわかるでしょう。

```
10 A=238
20 B=569
30 C=A+B
40 PRINT C
50 END
run
 807
```

なお、つぎのようにする、それぞれType mismatch というエラーメッセージが、表示されます。

```
10 A$=238
```

ストリングになっていません
ストリング変数には、クォーテーションマークで囲んでストリングして入れます

```
10 A="238"
```

これはストリングです
数値変数には、数値を入れるようにします

ストリング変数のときは、A＄＝〝２３８〞、数値変数のときはA＝２３８としなければなりません。

まえにPRINT　A＄；B＄；C＄といったように、セミコロンで区切ると、文字がつづいて表示する例をとりあげました。

では、PRINT　A＄，B＄，C＄といったように、「，」コンマで区切ると、どうなるでしょうか。

```
10 A$="イヌ"
20 B$="サル"
30 PRINT A$,B$
40 END
run
イヌ            サル
```

14字

コンマで区切ると、このように一定の間隔をあけて表示します。

ストリング変数とINPUT

クォーテーションマークがいらないストリング

INPUTは、つぎのプログラムのように、INPUTのあとに変数をともなっていて、その変数に読み込むためのデータを、？マークを表示して、私たちに求めてくるというものでした。

```
10 INPUT A,B      ←AとBのデータを読み込みなさい
20 S=A*B          ←A×Bを行って、その結果を左側のSに入れなさい
30 PRINT S        ←Sの中身を表示しなさい
40 END
run               実行させると？マークを表示します
? 25,15 RETURNキーを押します      データを入力します
 375              ←A×Bを行った結果
```

このINPUTのあとにつづく変数AとBを、ストリング変数A＄やB＄にかえて、ストリングを処理させることができます。

ストリングとは、文字の集まり（文字列）を、クォーテーションマークで囲んだものをいいます。

```
"ｱﾅﾀﾉﾄｼﾊ ｲｸﾂﾃﾞｽｶ"
"16ｻｲﾃﾞｽ"
"BOY AND GIRL"
```

しかし、パソコンがINPUTを実行して、？マークを表示してきて、その横に私たちが文字を入力するときは、クォーテーションマークで

囲む必要はありません。

```
run
? タロウデ゛ス
```

INPUTのときは、クォーテーションマークで囲まないで入力してもよいのです

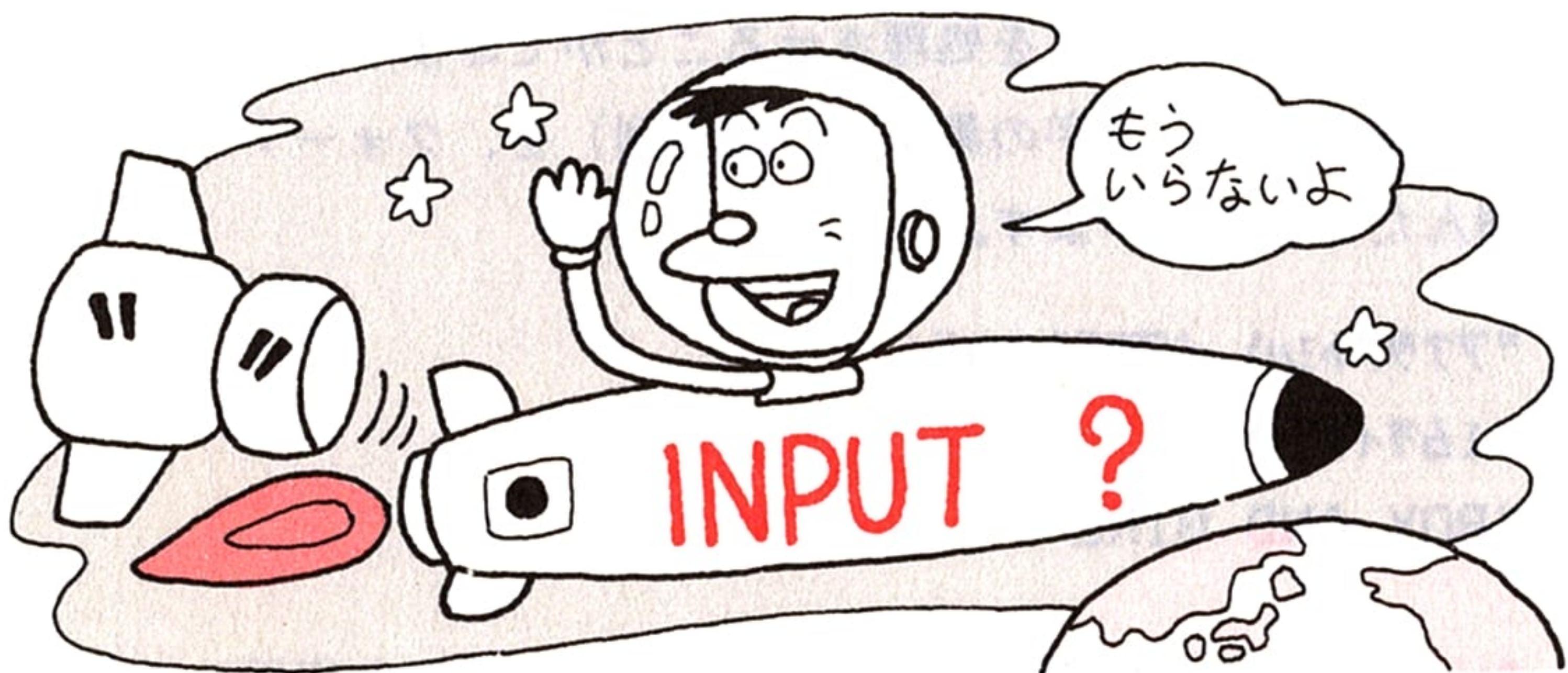
このように入力しても、パソコンは、クォーテーションマークで囲んだ文字と、おなじように扱って処理してくれます。

では、実際にプログラムを実行して、みてみましょう。

```
10 INPUT A$,B$
20 PRINT A$,B$
30 END
```

このプログラムをパソコンに入力して、RUNを入力すると、パソコンはINPUT　A＄，B＄を実行して、A＄とB＄に入るデータとしてのストリングを、？マークを表示して求めてきます。

```
run
? タロウ,ハナコ
```

A＄にはタロウ、B＄にはハナコが入るように入力しました。そし

て RETURN キーを押すと、A＄にタロウ、B＄にハナコが読みとられます。

```
20 PRINT A$,B$
30 END
run
? タロウ,ハナコ
```

そしてつぎに、A＄のタロウとB＄のハナコは、行番号20のPRINT　A＄、B＄に送り込まれます。

```
10 INPUT A$,B$
         タロウ ハナコ
20 PRINT A$,B$
```

PRINTは、表示しなさいということです。PRINT　A＄、B＄が実行されると、つぎのようにA＄の中身タロウと、B＄の中身ハナコがディスプレイに表示されます。

```
run
? タロウ,ハナコ
タロウ                ハナコ
```

102頁のプログラムは、通常の変数Rと、ストリング変数A＄を組み合せて使ったものです。

```
10 INPUT A$,R        ← INPUTで、A$にストリング、Rに数値を読み込みます
20 X=3.14*R*R        ← 3.14×半径の2乗。円の面積を求めます 計算結果は左側のXに入ります
30 PRINT A$;X        ← Xの中身は計算結果 A$の中身はストリングです
40 END

run
? ｴﾝﾉﾒﾝｾｷﾊ,5         ← 入力したデータ。エンノメンセキハはA$へ、5はRに入ります
ｴﾝﾉﾒﾝｾｷﾊ 78.5        ← 実行結果です
```

行番号10のINPUT　A＄，RのA＄は、ストリング変数、Rは変数です。A＄にはストリングを、Rには数値を入力します。

行番号10を実行すると、パソコンは？マークを表示してきます。

うえのプログラムは、円の面積を求めるプログラムですから、A＄にはエンノメンセキハを入力し、Rには5を入力しました。

Rの値5は、行番号20のRとRに送り込まれて、3.14×5×5が計算されます。計算結果の78.5は、＝の左側のXに送り込まれてから、行番号30のXに送られます。行番号30のA＄には、行番号10のA＄で読み込まれた、エンノメンセキハが送り込まれます。

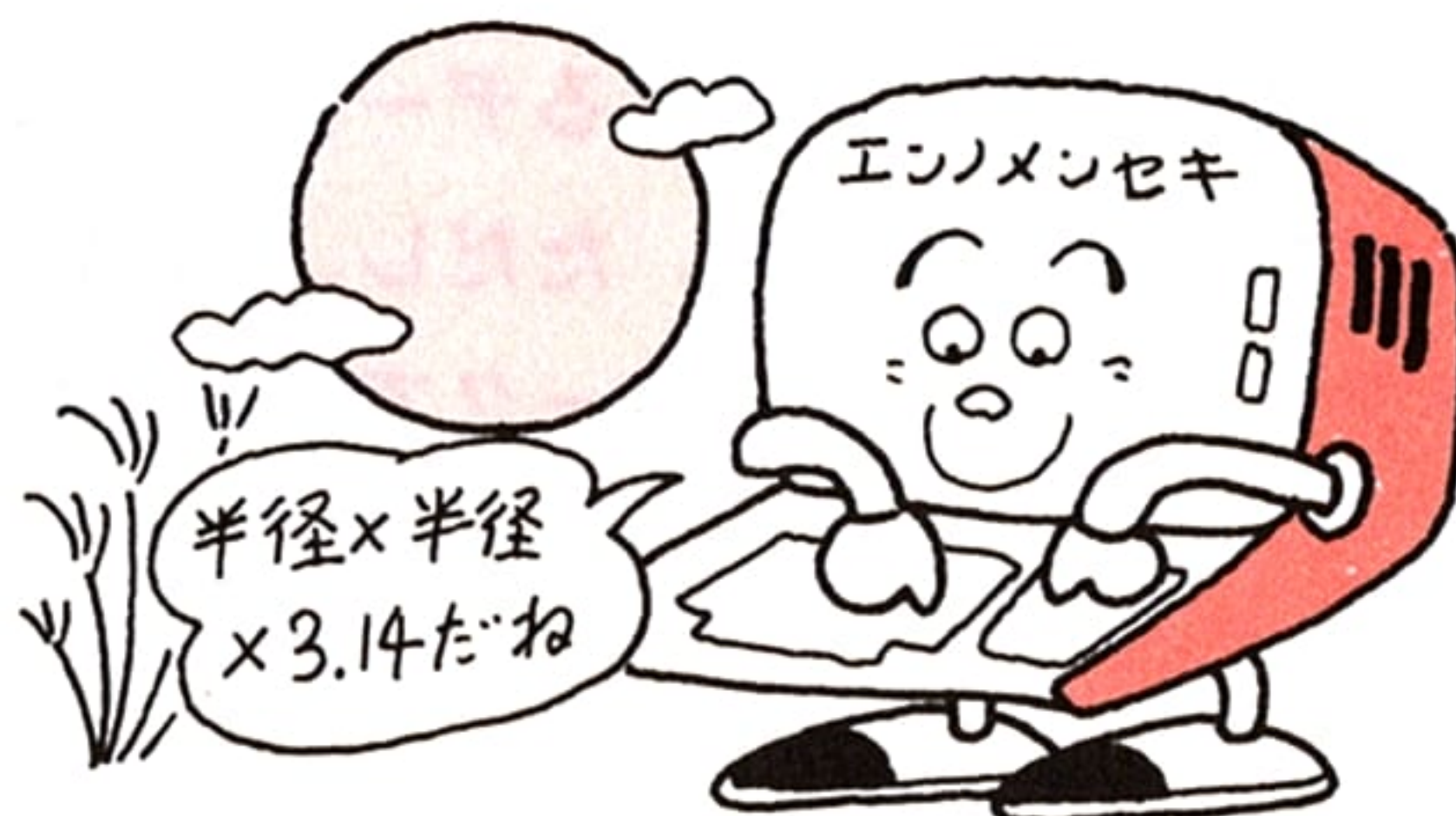
行番号30のPRINT　A＄；Xは、A＄のエンノメンセキハを表示したあと、つづけてXの78.5を表示しなさいということです。run のあとにある実行結果をみてもわかるように、ストリング変数と通常の変数を組み合せて使うと便利です。

では、通常の変数に読みとらせた数値を、ストリング変数に読みとらせたら、どうなるでしょうか。

```
10 INPUT A$,B$
20 S$=A$+B$
30 PRINT S$
40 END
```

パソコンは、INPUT　A＄、B＄を実行して、？マークを表示して、A＄、B＄のデータを要求してきましたから、1254と9327を入力してみます。A＄の1254とB＄の9327は、行番号20のA＄＋B＄に送られてプラスされ、その結果がS＄に入り、ついで行番号30のS＄に送り込まれて、PRINT　S＄で表示された結果が、おわりに示されている1254 9327です。

```
run
? 1254,9327   ← RETURNキーを押す
12549327      ← 求められた答
```

このようになった理由は、ストリング変数に読みとられた数値は、数値としての意味を失って、文字として扱われるためです。したがって、ストリング変数を使うときには、数値の扱いに十分に気をつけてください。

ストリング変数とREAD～DATA

READ ～ＤＡＴＡでもストリングが扱える

173頁でおはなしするREAD～DATAは、つぎのプログラムのようなものでした。つまりREADのあとにつづく変数に、DATAのあとにつづくデータを読みとらせて、処理するものでした。

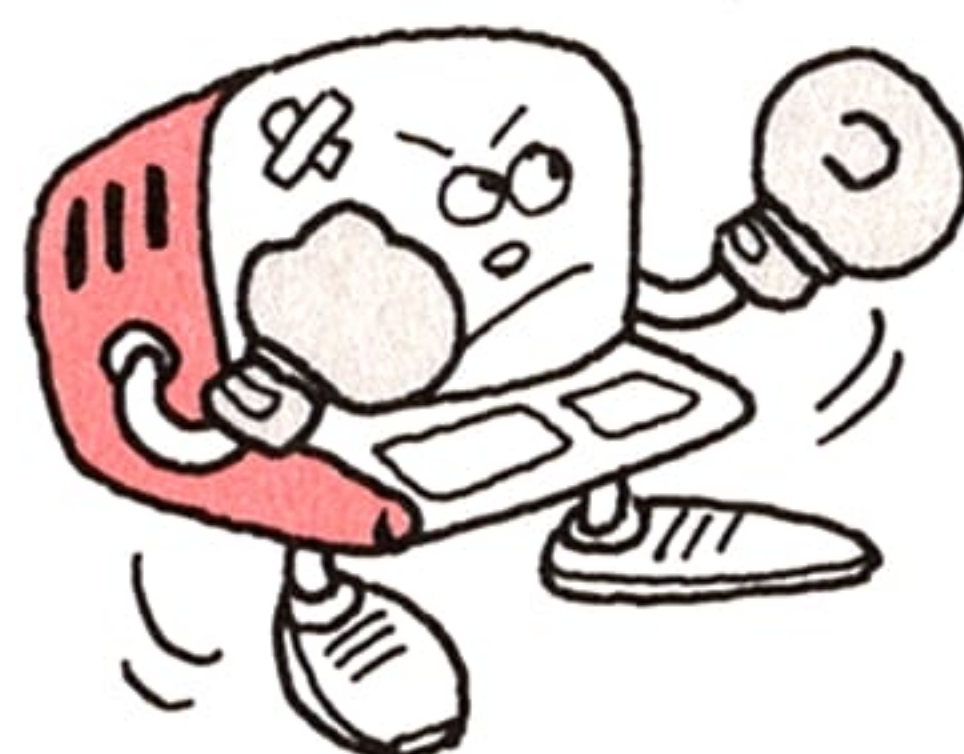
```
10 READ A,B
20 S=A*B/2
30 PRINT S
40 END
50 DATA 30,20
run
 300
```

行番号50のＤＡＴＡから、変数Ａに30、Ｂに20を読みとらせます

Ａの30とＢの20が送り込まれて、30×20÷２が行なわれ、その結果の 300は、左側のＳに入れられます

行番号20のＳから計算結果 300が送られてきて、ＰＲＩＮＴ　ＳでＳの中身 300が表示されます

うえのプログラムの、READのあとにつづくＡとかＢとかいった通常の変数を、ストリング変数Ａ＄、Ｂ＄にかえると、READ～DATAでも、ストリングを扱うことができます。

もちろん、READにつづく通常の変数を、ストリング変数にしたときは、DATAのあとに示されるデータも、数値ではなくて、ストリングにしなければなりません。ただし、READ～DATAのときは、ストリングをクォーテーションマークで囲む必要がありません。

```
10 READ A$,B$
20 PRINT A$,B$
30 DATA ｵﾄｺﾉｺ,ｵﾝﾅﾉｺ
40 END
```

行番号30のDATAのあとに置かれている、オトコノコ、オンナノコのように、READ～DATAの場合は、クォーテーションマークで囲まなくても、ストリングとして扱われることになっています。

このプログラムを実行させると、まずREAD　A＄、B＄が実行されます。READ　A＄、B＄は、DATAに示されているストリングを、ストリング変数A＄、B＄に読みとりなさいということです。そこで、オトコノコがA＄に、オンナノコがB＄に読みとられます。

行番号10のA＄とB＄に読みとられたオトコノコ、オンナノコは、つぎに、行番号20のPRINT　A＄、B＄のA＄とB＄に送り込まれます。

```
10 READ A$,B$
    (オトコノコ) (オンナノコ)
20 PRINT A$,B$
```

PRINT　A＄、B＄は、A＄とB＄の中身を表示しなさいということですから、A＄とB＄の中身が表示されます。

```
run
ｵﾄｺﾉｺ          ｵﾝﾅﾉｺ
```

では、DATAのあとに、数値を置いてみましょう。

```
10 READ A$,B$
20 C$=A$+B$
30 PRINT C$
40 END
50 50 DATA 125,305
```

この場合、行番号20でＡ＄＋Ｂ＄として、足し算の記号を使っていますが、果たして足し算は、行われるでしょうか。答えはノーです。なぜなら、ストリング変数に読み込まれた数値は、数値としての意味はなくなって、文字として扱われるからです。データの読みとられ方や処理の仕方は、まえとおなじです。

```
run
125305
```

この場合の＋の記号は、たんにつなげるという働きしかしていません。このようなことから、ストリング変数に、＋以外の四則演算記号＊、／、－などを使うと

Type mismacth

というエラーメッセージが、表示されることになります。

このように、数値はストリング変数に扱われると、数値の意味を失って単なる文字になってしまいますから、その扱い方には十分注意してください。

STR$(X)

STR$（X）は、変化する数値をストリングにする

数値をストリングにするには、数値を「〝」クォーテーションマークで囲めばよいわけです。

　　A＄＝〝4321〞

　　B＄＝〝97865〞

しかし、つぎのようなプログラムがあって、その計算結果をストリングにする場合、ひとつひとつクォーテーションマークで囲むわけにはいかず、困ってしまいます。

```
10 READ A
20 X=A*A
30 PRINT X
40 GOTO 10
50 END
60 DATA 10,25,9,8,28
```

10 READ A ← Aの値はDATAから読みとられます

20 X=A*A ← A×Aが行われて、その計算結果は左側のXに入ります

30 PRINT X ← Xの中身が表示されます

60 DATA 10,25,9,8,28 ← この値はREAD　AのAに読みとられます

このプログラムのX＝A＊AのXのように、Xの中身である数値が、

STR$ (X)

常に変化するようなときに、その数値をストリングにするのが、STR$（X）なのです。このSTR$（X）のXに入った数値が、たとえば、100であるとすると、その数値は“100”というストリングに変換されます。

では、X＝A＊AのXの数値がストリングになるように、STR$(X)を使ってみましょう。

```
run
 100
 625
 81
 64
 784
```

これは
"100"
"625"
"81"
"64"
"784"
です

STR$(X)で書きだされた、この数字は、ストリングであって、数値ではありません。つまり文字の集まりなのです。

では、なぜ、STR$(X)を使って、わざわざ数値をストリングに変換する必要があるのでしょうか。

パソコンの計算結果などの書きだし方をみるとわかるように、すべて、頭がそろっていて、1,000の桁も、100の桁も、10の桁もないといった書きだし方です。

私たちが、日常、数字を書くときは、必ず桁をあわせて書きます。

```
2,521
   65
  253
4,670
  890
```

このように、私たちが数字を書くのは、そのほうが、数字がみやすく、わかりやすいためです。

パソコンに桁をあわせて書きださせるためには、計算結果を数値から文字に変換し、そのうえで、桁をあわせるように処理しなければならないのです。

このように数値は、数値のままでは、いろいろな処理ができない場合が、多くあるので、数値をストリングに変換する、STR$(X)があるのです。

STOP

STOPは、プログラムの実行を中断させる

パソコンにプログラムを入力して、実行させている途中で、そのプログラムの実行を中止させるとき、私たちはキーボードのSTOPキーとCTRLキーを一緒に押します。

するとパソコンは、Break in（行番号）といったように、実行を中断したところの行番号を書きだして、実行を中断したというメッセージを表示します。

このSTOPを、キーボードから入力するのではなく、プログラムのなかに書き込んでおくことができます。

これがSTOPという、ことばです。

```
10 INPUT R
20 S=3.14*R*R
30 Y=R*R
40 STOP
50 PRINT "ｴﾝﾉﾒﾝｾｷ=";S
60 PRINT "ｾｲﾎｳｹｲﾉﾒﾝｾｷ=";Y
70 END
```


STOPは、プログラムのなかのどこに使ってもかまいませんし、また、いくつ使ってもかまいません。

パソコンは、プログラムのなかで、このストップに出合うと、STOPキーとCTRLキーを一緒に押したときとおなじように、Break in

（行番号）というメッセージを表示して、プログラムの実行を中止します。中止したといっても、END（おわり）のように、実行を終了してしまったわけではありません。ですからCONTという命令を入力すると、ふたたびプログラムを実行します。

まえのプログラムでは、行番号40にSTOPを使っています。

このプログラムをRUNさせると、行番10のINPUT　Rが実行されて、パソコンは、？マークを表示して、Rの値を求めてきます。

Rの値を20として入力し、RETURN キーを押すと、パソコンは行番号20の3.14×20×20という、円の面積を求める計算をし、その結果を左側のSに入れます。そして、行番号30に進んで、20×20という正方形の面積を求める計算をして、その結果を左側のYに入れます。

行番号40は、STOPですから、そこでBreak in 40を表示して、実行を中断します。

さて、Break in 40の下に、OKが表示されていないでしょうか。このOKは、「つぎの命令を待っている」ということを示すOKなのです。

したがって、CONTという命令を入力すると、ふたたび実行しますが、そのまえに、やってみたいことがあります。

PRINT S, Y

というPRINT命令を入力してみてください。

この場合は、直接命令ですから、PRINTの頭に行番号をつける必要はありません。RETURN キーを押すと、SとYに入っている計算結果を表示します。

```
print S,Y  (RETURNキーを押す)
 1256           400
```

パソコンは、実行を中断してはいますが、入力されたデータや、計算結果などを忘れてしまっているわけではありません。このような状態を「ファイルを閉じていない」と呼んでいます。

短いプログラムでは、ほとんど用いませんが、長いプログラムの場合は、ところどころにSTOPを使って、実行を中断させ、このようにして、中間のチェックをしたりします。

STOPは、ただたんに実行を中断させるためばかりでなく、このような役割を果たすものであることも、おぼえておいてください。

では、実行をふたたび行わせるCONTを入力してみましょう。CONTを入力すると、パソコンは、行番号50、60を実行して、

```
ｴﾝﾉﾒﾝｾｷ= 1256
ｾｲﾎｳｹｲﾉﾒﾝｾｷ= 400
```

を表示すると、つぎに行番号70のEND（おわり）を実行して、プログラムの実行を終了したという、OKを表示します。

ENDを実行すると、CONTを入力しても、OKを表示するだけです。

SPC(X)

SPC(X)は、空白をあけて文字を表示する

数字や文字などをディスプレイに表示させるとき、ディスプレイの左端から何字分か、空白をあけて表示させたり、また文字と文字の間、数字と数字の間を何字分か、空白をあけて表示させたりします。このようなとき、SPC（X）を使うと便利です。

SPCは、Spaceの略で、SPC（X）のXに、ある数を与えると、その数だけの空白を作りだします。

このSPC（X）は、つぎのようにPRINTといっしょに使います。

```
PRINT SPC(X) A
PRINT A SPC(X) B
```

PRINT SPC(X) A ← ディスプレイの左端からX字分空白をあけて表示します

PRINT A SPC(X) B ← 字と字の間に、X字の空白をあけて表示します

では、実際にプログラムでみてみましょう。

```
10 A$="ﾀﾛｳ,ﾊﾅｺ"
20 A=12345
30 PRINT SPC(5) A
40 PRINT SPC(5) A$
50 PRINT SPC(10) A$
60 PRINT SPC(10) A
70 END
```

ストリング変数A＄には、ストリングであるタロウとハナコが入り、変数Aには、12345 といった数値が入ります。したがって、行番号40

と50では、タロウとハナコが、行番号30と60では、12345 が、SPC（５）、SPC（10）で指定された数だけ、空白をあけて表示されます。

うえの実行結果をみると、文字の場合はきっちり５字分あけて表示されますが、数字の場合は、文字より一字よけいに、空白があけられます。これは、マイナスの符号のための、空白です。

```
10 A=123
20 B=456
30 PRINT A SPC(0) B
40 PRINT A SPC(1) B
50 PRINT A SPC(2) B
60 PRINT A SPC(3) B
run
 123   456 ←——SPC(0)のとき
 123    456 ←——SPC(1)ののき
 123     456 ←——SPC(2)のとき
 123      456 ←——SPC(3)のとき
```

「;」セミコロン

セミコロンは、文字などをつづけて表示する

78頁でおはなししたコンマは、PRINT文のなかで文字と文字の間に、また数字と数字の間に使うと、14字分という、一定の間隔をあけて、文字や数字を表示しました。

ところがセミコロンを、PRINT文のなかで文字や数字の間に使うと、コンマとは逆に、文字はつづけて表示され、数字は一字あけて表示されるようになります。

```
10 A$="ｲﾇ"
20 B$="ｻﾙ"
30 PRINT A$;B$
40 END
run
ｲﾇｻﾙ
```

つづけて表示されます

```
10 A=123
20 B=456
30 PRINT A;B
40 END
run
 123  456
```

一字アケテ表示されます

マイナス符号が入る場所です

「；」セミコロン

数字のまえには、マイナス符号が入る場所として、必ず一字分よけいにあいています。

さて、数字の場合は、セミコロンを使っても一字空白が作りだされるからよいのですが、文字の場合は、ぴったりとついてしまいます。これではみにくいので、何個か空白を作りたいときには、つぎのようにすればよいでしょう。

```
10 A$="イヌ□□"
20 B$="サル"
30 PRINT A$;B$
40 END
```

ストリングでは、
空白も文字としてかぞえられます

セミコロンそのものの働きには、手を加えることができませんから、「〃」クォーテーションマークのなかに空白をつくります。クォーテーションマークのなかの空白は、一字分なら一字、二字分なら二字としてかぞえられますから、とりたいだけの空白を作ることができます。うえのプログラムでは、〝イヌ　　〟の横に二字分の空白をとりました。これは〝　　サル〟のまえにしても、おなじです。RUNさせると、つぎのようになります。

```
run
イヌ  サル
```

ストリングのなかに空白をとっただけ、
空白を作りだします

添え字つき変数とDIM

添え字つき変数は、記憶場所をいくつも確保する

ふつう、変数というと、A、B、Cといったようにアルファベット1文字か、A1、B2、C3といったアルファベット1文字にひとつ数字がついたものをいいます。

これら変数のあとに、カッコで囲んだ数字の部分がつくと、添え字つき変数と呼ばれる変数になります。

A(5)　B(10)　C3(8)　D9(23)

添え字

これは、カッコのなかの数字の部分を、添え字と呼ぶところからきています。

カッコのなかの添え字は、必ずしも、5とか10とかのような数値が入るとは、かぎりません。Iとかアとかいったように、アルファベット1字でできている変数や、数式が入ることもあります。つまり、

A(I)　T(X)　S(B−C)　Y(5*I)

といった具合です。

ただ、どんな場合でも、このカッコのなかの添え字は、正の整数でなければならないことになっています。たとえば、カッコのなかの添え字が、A(−8)といったように負の整数であると、パソコンは、プログラムを実行したとき、

Illegal function call

といったエラーメッセージを表示します。Illegal function callとは、ステートメントの機能の呼び方が間違っているという意味です。

添え字つき変数とDIM

ところで、添え字つき変数とは、どんな働きをするのでしょうか。

たとえば、A（7）といった添え字つき変数の場合では、Aという変数のために、A（0）からA（7）までの、8個の記憶場所を、まえもってメモリのなかに確保してしまうといった、働きをします。

もちろん、添え字つき変数は、単独では使えません。つぎのように、DIMということばを、その頭につけて使います。

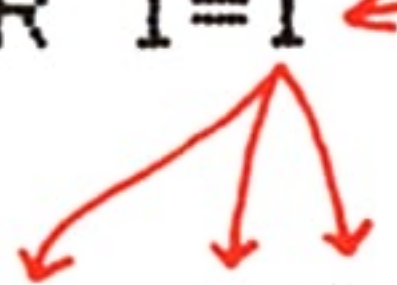
```
DIM A(7)
```

こうすることではじめて、Aという変数のための記憶場所をまえもって、メモリのなかに確保することが、できるようになります。

この場合、DIM　A（7）なら記憶場所の確保は7個と考えがちですが、パソコンの多くは、0からはじめられるので、つぎの図のように、8個の場所が確保されるのです。DIM　A（10）なら11個ということになるわけです。

A(0)	A(1)	A(2)	A(3)	A(4)	A(5)	A(6)
A(7)						

では、Aのために確保された記憶場所は、どう使われるのでしょうか。つぎのプログラムで、みてみましょう。

```
10 DIM A(7)
20 FOR I=1 TO 7
30 A(I)=I*I
40 NEXT I
50 END
```

まず、行番号10DIM　A（7）で、メモリのなかに変数Aの記憶場所を8個確保しました。前頁の図のようになります。

行番号20と40にFOR～NEXTが使われています。このプログラムのFOR～NEXTは、FOR　I＝1　TO　7ですから、Iが1から7になるまで、繰り返し処理しなさいということです。

そこでまず、I＝1が実行されます。I＝1は、Iの中身が1であるということですから、Iの中身の1が行番号30のすべてのIに送り込まれます。

```
20 FOR I=1
30 A(I)=I*I
```


したがって、行番号30は、つぎのようになるわけです。

30　A（1）＝1×1

そして、1×1が行われ、その計算結果の1は、＝の左側のA（1）に入れられます。つまりこのA（1）は、行番号10のDIM　A（7）で、A（0）からA（7）まで、メモリのなかに記憶場所が確保されていますから、その確保されているA（1）に入れられることになります。120頁の図のようにです。

行番号40は、NEXT　Iです。ここでIの1は、1プラスされて、2になります。この2は行番号10のIに送られます。したがって今度は、

I＝2になり、このIの中身2は、行番号30のすべてのIに送り込まれます。

ということは、A（2）＝2×2ということになります。2×2の計算結果の4は、＝の左側のA（2）に入れられます。つまり、これ

A(0)	A(1)	A(2)	A(3)	A(4)	A(5)
	1	4	9	16	25
A(6)	A(7)				
36	49				

も、一番最初とおなじように、メモリのなかに確保されている記憶場所A（2）に入れられることになります。

行番号40のNEXT Iで、Iの中身2は、おなじように1プラスされて、3になります。そして、行番号10のIに送られます。したがって、Iは3になって、また行番号30のすべてのIに送り込まれます。この

ようなことをＩが７になるまで繰り返して、行番号30のＩ＊Ｉの計算が行われ、その計算結果は、Ａ（Ｉ）のＩに示された記憶場所に前頁の図のように入れられます。この場合、Ａ（１）からはじまりますから、Ａ（０）は確保されていても、データは入りません。あいたままになっています。

さて、データをDIM　Ａ（７）で、メモリのなかに確保した記憶場所に入れました。入れるというだけでは、なんの意味もありません。そこで、Ａ（１）からＡ（７）までに記憶されているデータを引きだして、計算してみることにしましょう。

計算させるときは、つぎのように、それぞれデータが記憶されている場所を、四則演算記号を使ってつなげばよいのです。

```
10 DIM A(7)
20 FOR I=1 TO 7
30 A(I)=I*I
40 NEXT I
50 PRINT A(5)/A(2)
60 PRINT A(3)*A(6)-A(7)
70 PRINT A(2)+A(3)+A(4)
80 END
run
 6.25
 275
 29
```

（行10〜40）データをＤＩＭ　Ａ(７)で確保した記憶場所にしまいます　まえのプログラムと同じです

（行50〜80）データが、しまわれた記憶場所を四則演算記号でつないで計算させます。

このようにDIMと添え字つき変数を使って、メモリのなかに記憶場所を確保して、その確保した記憶場所にデータをしまうと、いろいろに、そのデータを利用することができるようになります。

添え字つきストリグ変数とDIM

添え字つきストリング変数は、文字列を記憶する場所を予約する

ストリング変数は、A、B、C、A1、B2といった変数に「$」ドルマークがついたものでした。

A$　B$　C3$　D8$

このストリング変数に、カッコでくくった添え字の部分がつくと、添え字つきストリング変数となります。

A$（10）　B$（15）　C3$（25）

ストリング変数は、数値ではなく、「〃」クォーテーションマークで囲まれた文字列（ストリング）を扱うものでしたが、添え字つきストリング変数も、数値ではなく、文字列を扱います。

文字列（ストリング）とは、

〝オトコノコ　オンナノコ〟〝ヨイコ　ワルイコ　フツウノコ〟というものです。

ただ、添え字つき変数は、文字列を扱うということでは、ストリング変数とおなじですが、その働きはまったく違います。添え字つきストリング変数の働きは、まえの項でおはなしした添え字つき変数とおなじです。

添え字つき変数は、A（5）、B（I）、C（I＊3）といった形で、これにDIMをつけてプログラムのなかに使うと、AならAという変数の記憶場所を、カッコでくくった添え字の数だけ、あらかじめメモリのなかに確保してしまうというものです。

この添え字つき変数とおなじように、添え字つきストリング変数も、A＄（10）、B＄（15）、C 3＄（25）といった形に、DIMをつけて、プログラムのなかに使います。たとえば、

DIM　A＄（10）

というようにすると、A＄というストリング変数の記憶場所を、まえもって、メモリのなかに確保してしまいます。

もちろん、添え字つき変数と、添え字つきストリング変数の働きはおなじであっても、添え字つき変数は数値を扱い、添え字つきストリング変数は、文字列を扱うといった相違があることを、おぼえておいてください。

ところで、さきに添え字つきストリング変数として、A＄(10)、B＄（15）といったように、カッコでくくった添え字の部分が、数値のものだけをとりあげましたが、必ずしも、添え字が、数値でなければならないというわけではありません。

A＄（I）　B＄（X）　C 3＄（B＊C）

といったように、添え字が変数であっても、計算式であってもかまいません。ただし、添え字が負の整数になるようだと、

Illegal function call

ステートメントの機能の呼び方が間違っているというエラーメッセージが表示されます。したがって、添え字は、必ず正の整数になるようにしなければなりません。

さて、つぎのプログラムが、添え字つきストリング変数を使ったプログラムです。

```
10 DIM A$(10)
20 FOR I=1 TO 10
30 READ A$(I)
40 NEXT I
50 DATA ナゴヤ,トウキョウ,オオサカ,キョウト,フクオカ
60 DATA サッポロ,センダイ,モリオカ,フクシマ,ヤマガタ
```

このプログラムを入力して、RUNを入力すると、まずパソコンは、行番号10を実行します。この場合は、DIM　A＄（10）ですから、A＄の記憶場所を、つぎの図のようにメモリのなかに確保します。

A＄(0)	A＄(1)	A＄(2)	A＄(3)	A＄(4)	A＄(5)	A＄(6)
A＄(7)	A＄(8)	A＄(9)	A＄(10)			

このように、メモリのなかにA＄の記憶場所を確保すると、パソコンは、行番号20の実行に移ります。行番号20は、FOR　I＝1　TO　10です。このステートメントは、行番号40のNEXTと対になっていて、Iが1から10になるまで、処理しなさいということです。なにを処理するのかというと、行番号20と40の間にはさまれている行番号30のREAD　A＄（I）を処理するのです。

行番号30のREAD　A＄（I）は、A＄（I）に、行番号60と70のDATAに示されているデータを読みとりなさいということです。READもDATAと、対になっています。

では、どんなふうに読みとられていくのか、順を追ってみましょう。

まずFOR　I＝1　TO　10は、Iが1からはじめてですから、Iの中身は1となります。このIの中身1は、READ　A＄（I）のI

に入ります。つまりＡ＄（１）となるわけです。それと同時に、READを働かせて、行番号60のDATAから、一番最初のデータであるナゴヤを、Ａ＄（１）に読みとらせます。

```
30 READ A$(I)

50 DATA ナゴヤ,トウキョウ,オオサカ,キョウト,フクオカ
```

READ　Ａ＄（１）のＡ＄（１）は、DIM　Ａ＄（10）で、すでにメモリのなかに記憶場所が確保されていますから、読みとられたナゴヤは、下の図のように、そのなかに記憶されます。

Ａ＄(０)	Ａ＄(１)	Ａ＄(２)	Ａ＄(３)	Ａ＄(４)	Ａ＄(５)	Ａ＄(６)
	ナゴヤ	トウキョウ	オオサカ	キョウト	フクオカ	サッポロ
Ａ＄(７)	Ａ＄(８)	Ａ＄(９)	Ａ＄(10)			
センダイ	モリオカ	フクシマ	ヤマガタ			

Ａ＄（１）の記憶場所にナゴヤを読みとると、パソコンは、行番号40　NEXT　Ｉを実行します。NEXT　Ｉは、つぎのＩの値を作るところで、ここでＩの中身１に１をたして２にします。そして、行番号20に戻ります。こんどは、Ｉの中身は２ですから、READ　Ａ＄（１）は、Ａ＄（２）になって、行番号60のDATAから、２番目のデータであるトウキョウを読みとります。

```
30 READ A$(I)

50 DATA ナゴヤ,トウキョウ,オオサカ,キョウト,フクオカ
```

そして、Ａ＄（２）の記憶場所に記憶します。このようなことを10回繰り返して、行番号60と70のDATAに示されているデータを全部、確保した記憶場所に記憶します。

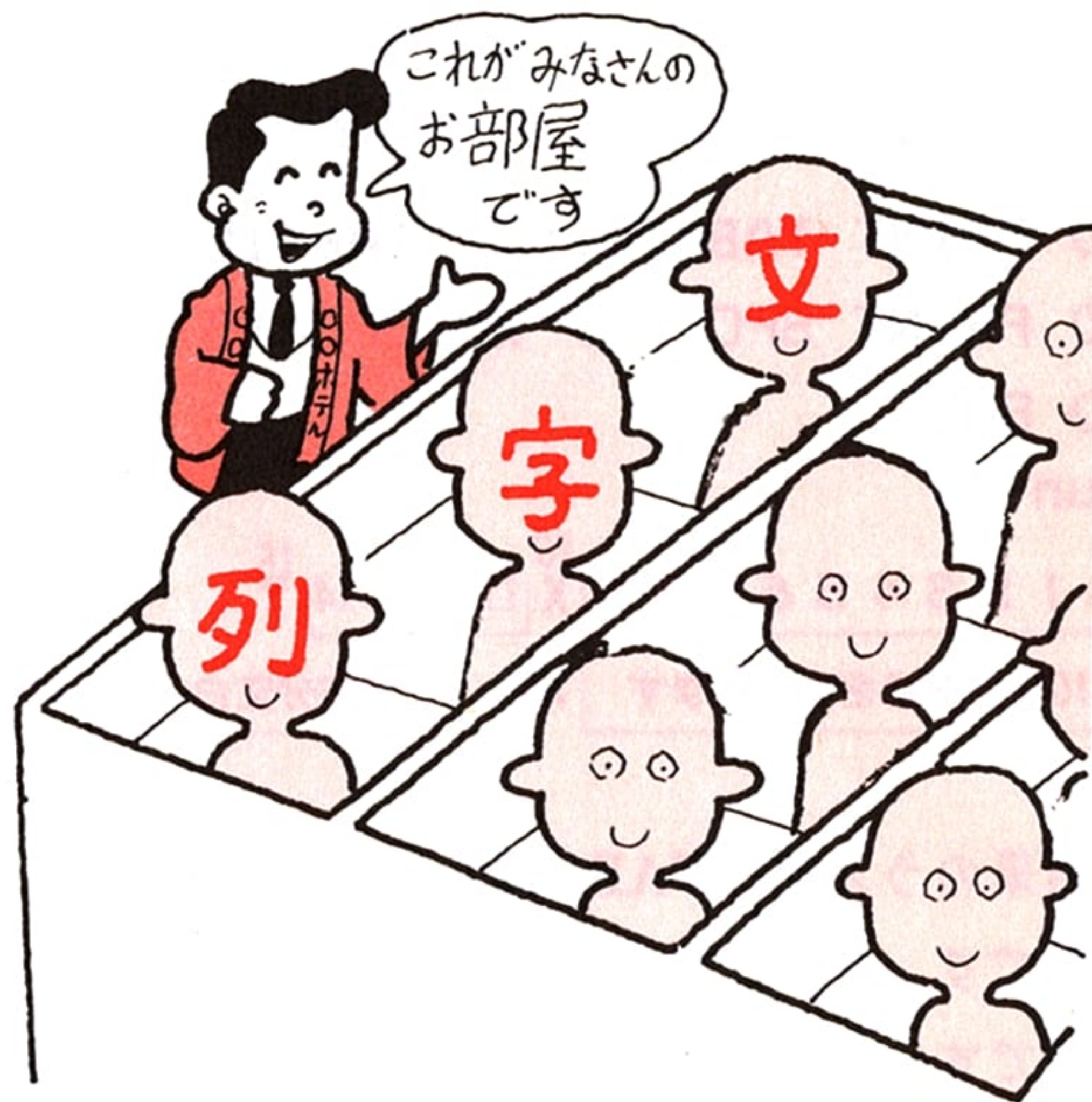
さて、このように、文字列（ストリング）を、DIM　Ａ＄（10）で確保したメモリの記憶場所に記憶させました。では、文字列を記憶させて、なにをしたらよいのでしょうか。これを利用する方法はいろい

ろありますが、ここでは、アイウエオ順に並び換える方法をとりあげておくことにしましょう。つぎのプログラムが、文字列を、アイウエオ順に並び換えるプログラムです。

このプログラムはDIM　A＄(15)として、A＄のために15の記憶場所を確保するようにしました。READ　A＄（I）で読みとるデータは、DATAに示されている15の都市名です。このプログラムを入力して、RUNしてみると、DATAに示されている都市名が、きちんとアイウエオ順に並び換えられて、表示されます。

```
10 DIM A$(15)
20 FOR I=1 TO 15
30 READ A$(I)
40 NEXT I
50 S=1
60 FOR X=1 TO 15
70 IF A$(X-1)=<A$(X) THEN 120
80 W$=A$(X)
90 A$(X)=A$(X-1)
100 A$(X-1)=W$
110 S=0
120 NEXT X
130 IF S=1 THEN 150
140 GOTO 50
150 FOR J=1 TO 15
160 PRINT A$(J)
170 NEXT J
180 DATA ｻｯﾎﾟﾛ,ﾊｺﾀﾞﾃ,ﾓﾘｵｶ,ｾﾝﾀﾞｲ,ｱｷﾀ
190 DATA ﾆｲｶﾞﾀ,ﾔﾏｶﾞﾀ,ﾌｸｼﾏ,ﾄｳｷｮｳ,ﾅｺﾞﾔ
200 DATA ｵｵｻｶ,ｷｮｳﾄ,ﾌｸｵｶ,ﾅｶﾞｻｷ,ﾀｶﾏﾂ
```

このプログラムの実行結果はつぎのとおりです。

```
run
ｱｷﾀ
ｵｵｻｶ
ｷｮｳﾄ
ｻｯﾎﾟﾛ
ｾﾝﾀﾞｲ
ﾀｶﾏﾂ
ﾄｳｷｮｳ
ﾅｶﾞｻｷ
ﾅｺﾞﾔ
ﾆｲｶﾞﾀ
ﾊｺﾀﾞﾃ
ﾌｸｵｶ
ﾌｸｼﾏ
ﾓﾘｵｶ
ﾔﾏｶﾞﾀ
```

TAB(X)

TAB(X)は、文字などを表示する位置を指定する

113頁でおはなししたSPC（X）と、TAB（X）は、よく比較されます。なぜならSPCとTABは、間違って使われやすいからです。

SPC（X）は、Xに与えられた数だけ、空白を作りました。SPC（10）なら、10個の空白を作りだしたわけです。つまり、SPC（X）のXに与えられた数は、作りだす空白の個数を表わしているわけです。

ところが、TAB（X）のXに与えられる数は、空白の個数を表わしているのではなくて、桁を表わしているのです。つまり、TAB（10）なら、10桁目に表示しなさいということになります。桁の番号は、左端から0、1、2、3といったようにかぞえていきます。

実際に表示させて、まず、その相違をみてみることにしましょう。

```
10 PRINT TAB(10)"X" TAB(15)"Y"
20 PRINT SPC(10)"X" SPC(15)"Y"
30 END
run
          X    Y
          X               Y
```


実行結果のうえが、TAB（10）によるXと、TAB（15）によるYの表示位置です。下が、SPC（10）によるXと、SPC（15）によるYの表示位置です。Xの表示位置は、ちょうどおなじ位置になるので、わかりにくいかもしれませんが、TAB（10）のXは、10桁目の位置に、

SPC (10) のXは、10個の空白をあけて、それぞれ表示されているのです。これが、もっともわかりやすい形で表示されているのが、Yの表示です。

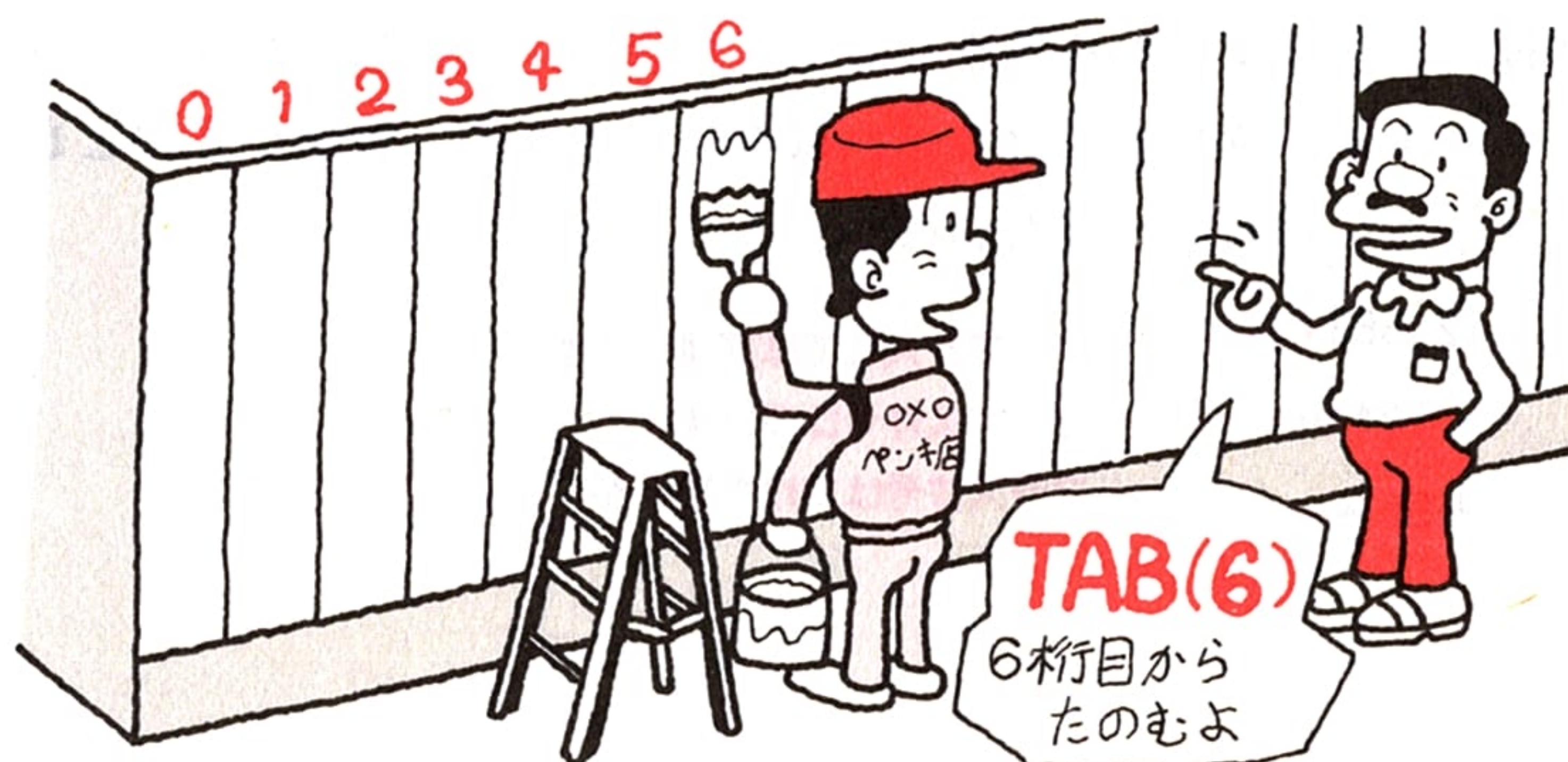
TAB (15)〝Y〟、SPC (15)〝Y〟は、カッコのなかの数はおなじですが、TAB (15) の15が、15桁目を示しているところから、15桁目の位置にYが表示されます。それに対して、SPC(15)の15は、Xの位置より15の空白を作りだして表示しなさいということですから、15の空白のあとにYが表示されます。

TAB (X) は、空白を作りだすことには違いありませんが、むしろ、桁を指定して、表示位置を決めるととらえておいたほうが、間違えないかもしれません。

なお、つぎのように、あとにつづくTAB (X) のXの数が、まえのTAB (X) のXの数より小さいと、正しく実行されません。

```
PRINT TAB(20)"X" TAB(10)"Y"
```

このようにすると
正しく実行されない

またTAB (X) は、必ずPRINTといっしょに使いますから、PRINT TAB (X)といっしょにして、おぼえておくほうがよいでしょう。

定　数

定数とは、決まっている数値

定数には、数値定数と文字定数の 2 つがあります。

ふつう、定数というときは、数値定数のことです。

文字定数は、ストリングとか、文字列などと呼ばれるのが、ふつうです。この本でも、文字定数をストリングとして、93頁でおはなししています。

数値定数などというと、むずかしく聞こえるかもしれませんが、プログラムのなかで使う、具体的な数値のことです。つまり私たちが、日常使っている、10進数の数のことです。

```
1234
−567
```

小数点をともなわない、この数値定数を、整数形式と呼んでいます。私たちは、日常、千の単位には「，」をつけますが、コンマを入れると間違いになりますから気をつけてください。また数値が負のときは、−（マイナス）符号をつけなければなりませんが、数値が正のときは＋（プラス）符号はつけないのが、ふつうです。

```
12,563 ←── コンマを入れては間違いです
−124   ←── マイナス符号はつけます
+56    ←── プラス符号はつけないのが、ふつうです
```

12.357

−6.654

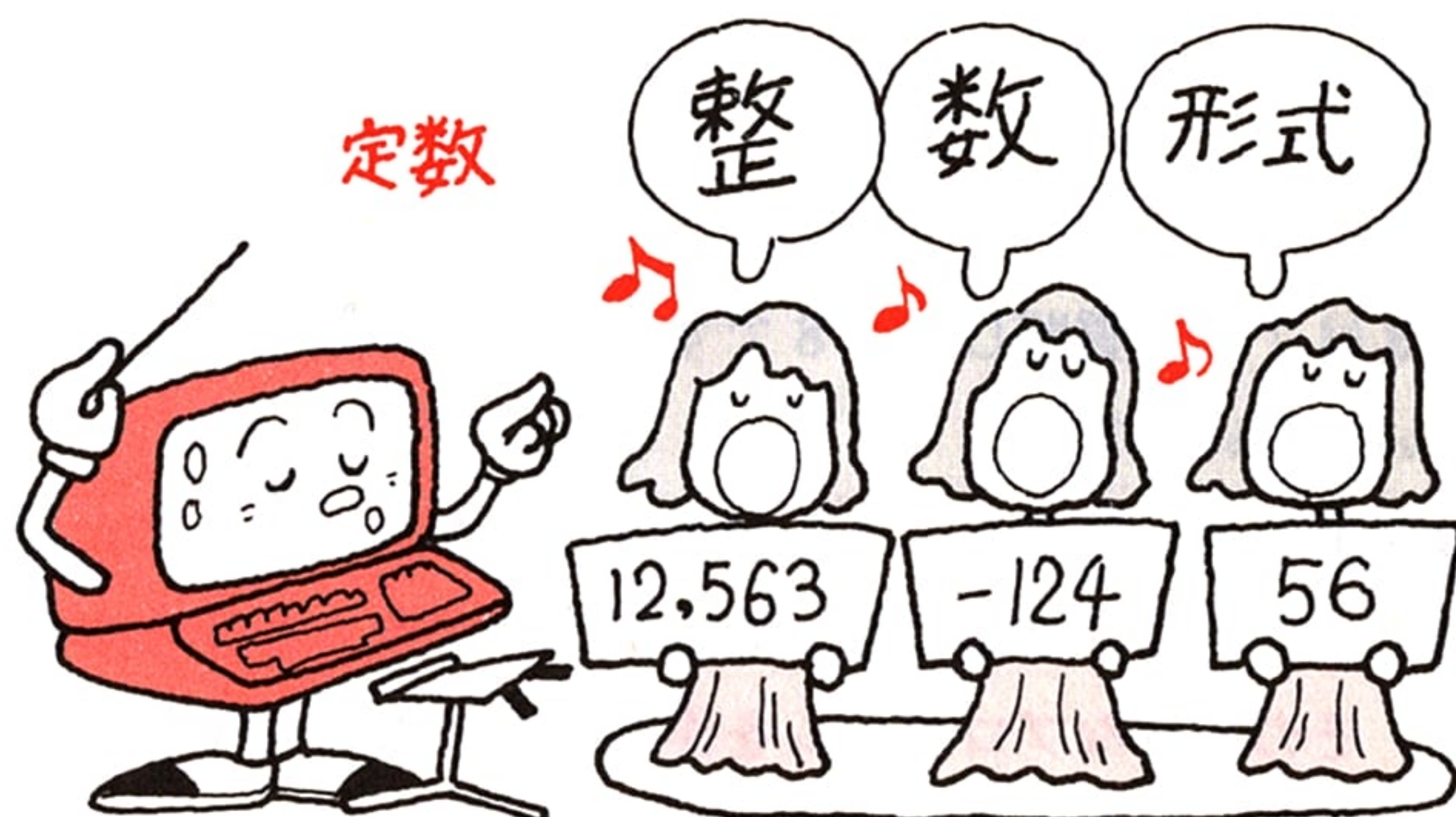
3.7

うえのように、小数点をともなった数値定数を、実数形式といいます。この場合も、数値が負のときは、−（マイナス）符号をつけなければなりませんが、数値が正のときは、＋（プラス）符号はつけないのが、ふつうです。

また、よくパソコンで計算などをしていると、つぎのように、Ｅをともなった数値がでてくることがあります。

23.766E−5(23.766×10^{-5}＝0.00023766)

1234E5(1234×10^{5}＝123400000)

これを指数形式と呼んでいます。これは、非常に大きい数値とか、非常に小さい数値を表わすのに用いられるものです。Ｅ−５とか、Ｅ５などが、指数部です。この指数部は、10のべき乗を表していて、Ｅのまえに書かれている数値に掛けます。つまりうえのカッコのなかに示したようになります。この指数部もＥのつぎの数値が負のときは、Ｅ−５といったようにマイナス符号がつきますが、正のときは、プラスの符号を書かないのが、ふつうです。

流れ図

流れ図は、プログラムを作るときの設計図

パソコンに仕事をさせるのは、プログラムであるということは、いまでは、もう常識となりました。

そのプログラムは、パソコンに仕事をさせるために、パソコンの手とり足とり教えるというぐらいに、こと細かに書かれています。

たとえば、パソコンに足し算をさせるという、簡単なプログラムであっても、足し算に必要な手順を、ひとつひとつ命令していかなければならないのです。

```
10 A=5
20 B=3
30 S=A+B
40 PRINT S
50 END
run
 8
```

「5＋3は？」と質問して、「8です」と答えが返ってくるというわけには、いかないのです。

こんな簡単な足し算でも、こんな具合ですから、もっと大きな複雑な仕事をパソコンにさせるには、大変なプログラムを作らなければなりません。そのようなプログラムを作るとき、なんの準備もなく、いきなりとりかかってもうまくいきません。

そのようなとき、つぎのような、プログラムの作り方をします。

① マイコンに行わせる仕事を、はっきり決める

② まとまったひとつの仕事を、なるべく単純な動作に分解する

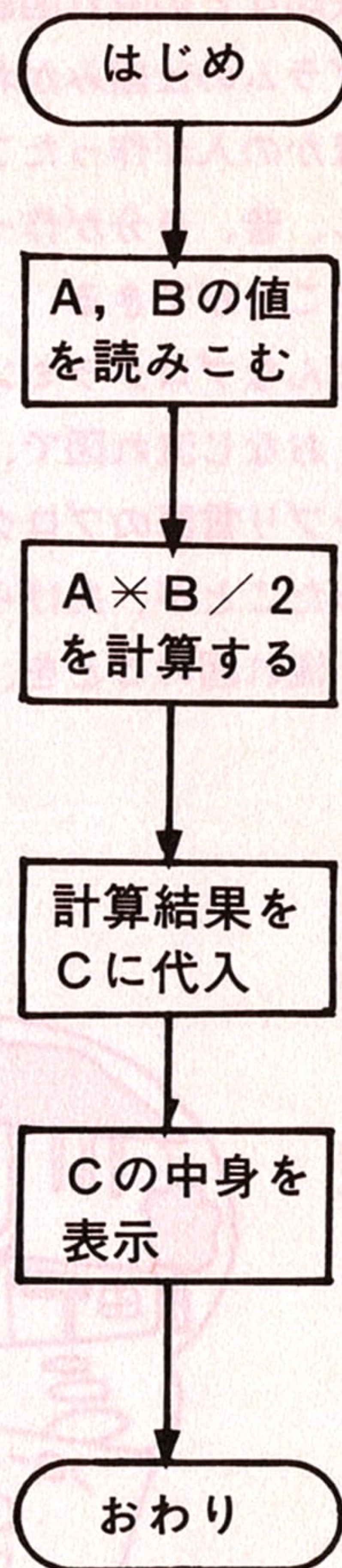
③ なるべく単純な動作に分解したため、バラバラになってしまった動作を、こんどは順序よく並べ直して、組み合せる。

④ 並べ直して、組み合せた結果をもとにして、プログラムを書きあげる

流れ図とは、③の動作を並べ組み合せた一連の手順を、わかりやすく、表現する手段です。

家を建てるときに、設計図が必要です。設計図を作らないで、家を建てるということは、まず、ありえません。

流れ図は、ちょうど、この家を建てるときの設計図にあたります。

つまり、プログラムを作るときは、プログラムを作るための設計図である流れ図を作って、それにもとづいて書きあげていくわけで

す。

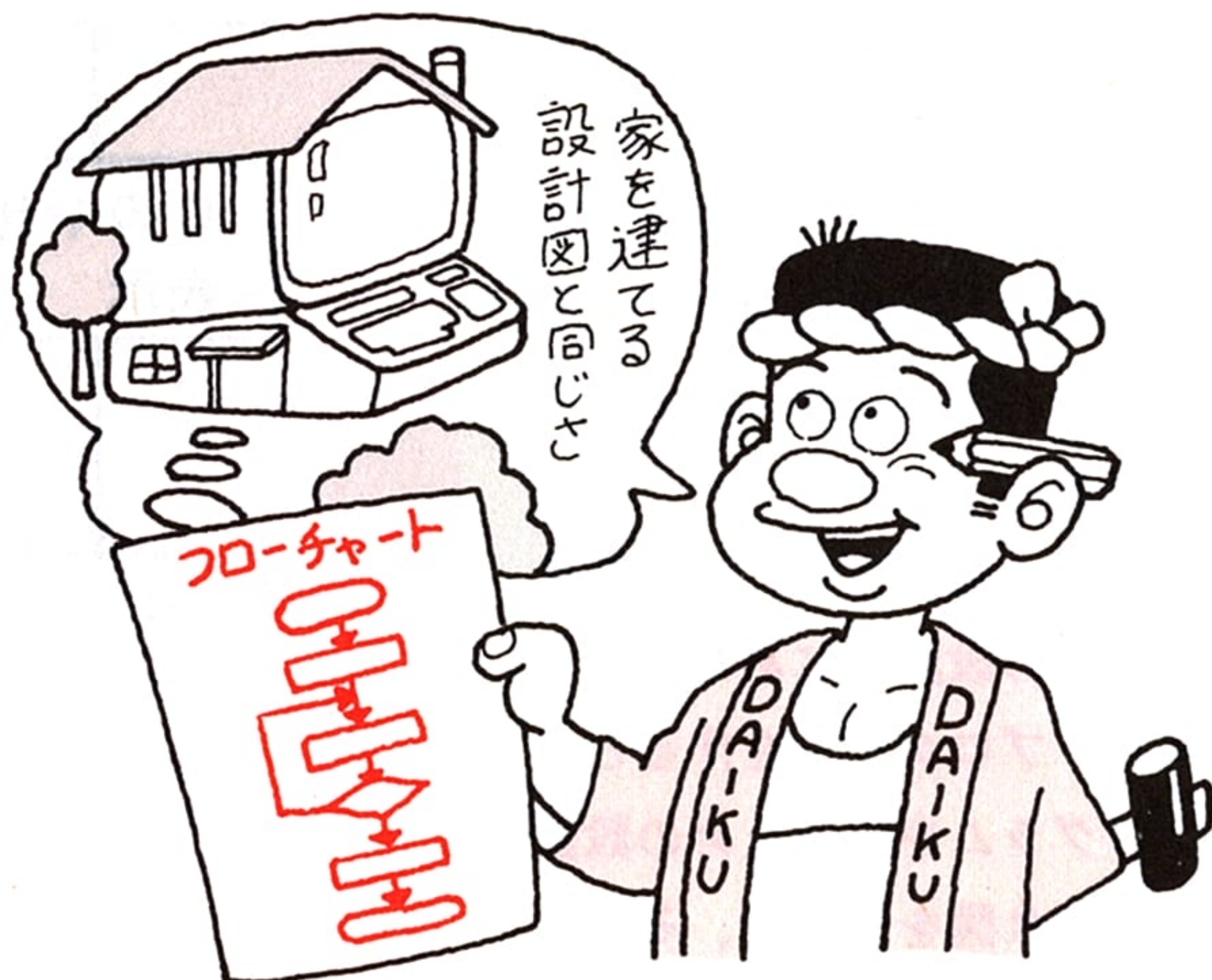
流れ図の特長は、

① 矢印などの流れ図記号を使っているので、一目みただけで、プログラムの仕組みがわかる

② ほかの人が作ったプログラムをみても、どうなっているかわかるし、昔、自分が作ったプログラムをみても、その仕組みを思いだすことができる

③ どんなプログラミング言語を使っても、流れ図はかわらないので、おなじ流れ図で、ベーシックのプログラムでも、機械語やアセンブリ言語のプログラムでも作ることができる

といったことが、あげられましょう。

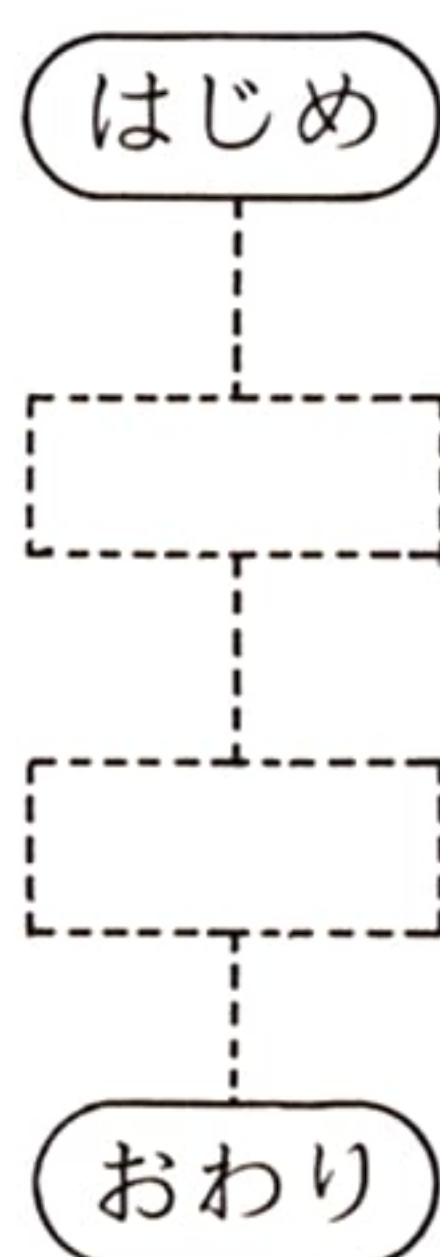
なお、流れ図のことを、フローチャートともいいます。

流れ図記号

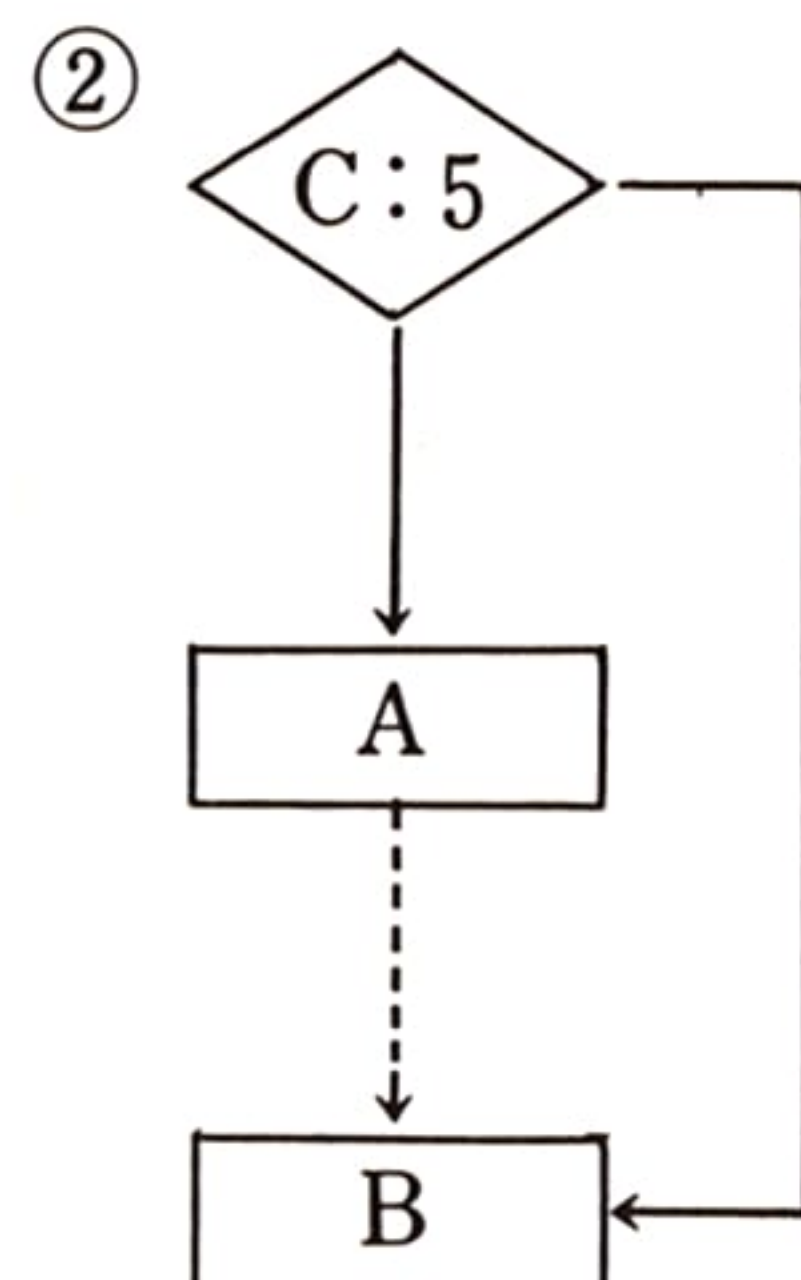
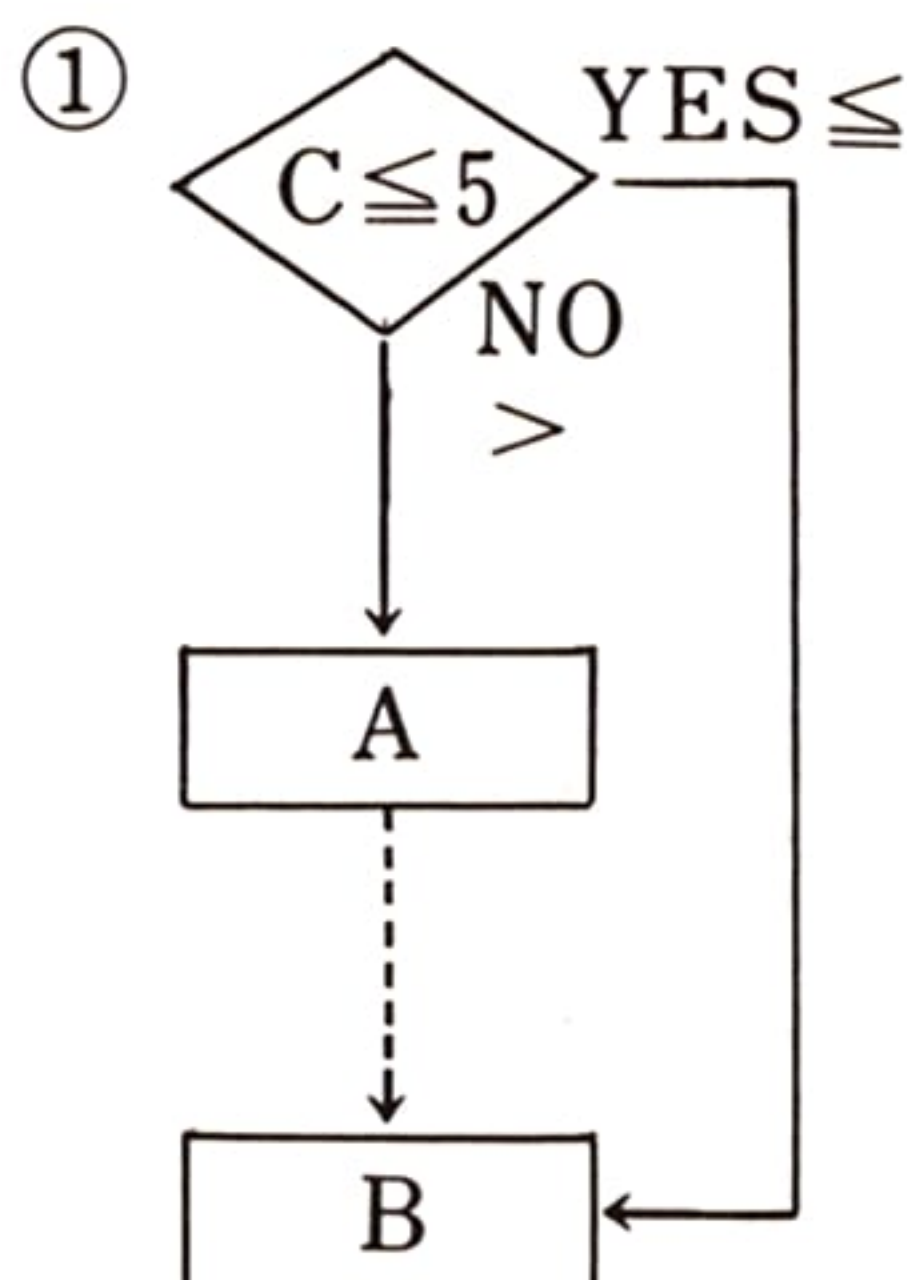
流れ図は、決められた記号を使って書く

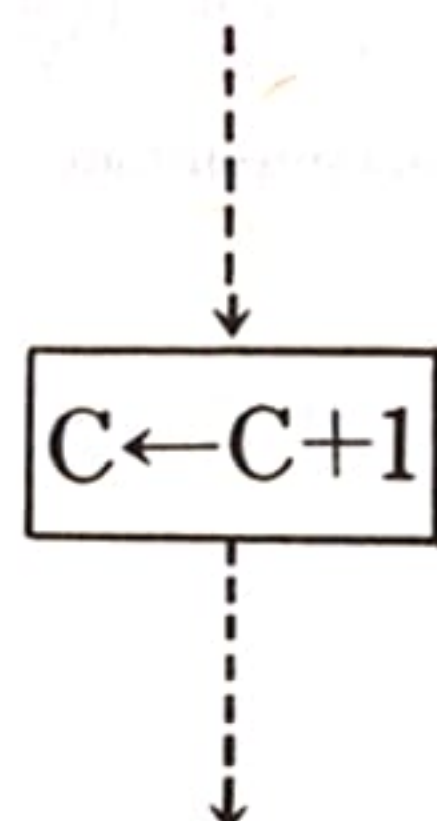
まえの項でおはなしした流れ図を書くときは、決められた記号を使って書きます。この決められた記号を、流れ図記号といいます。

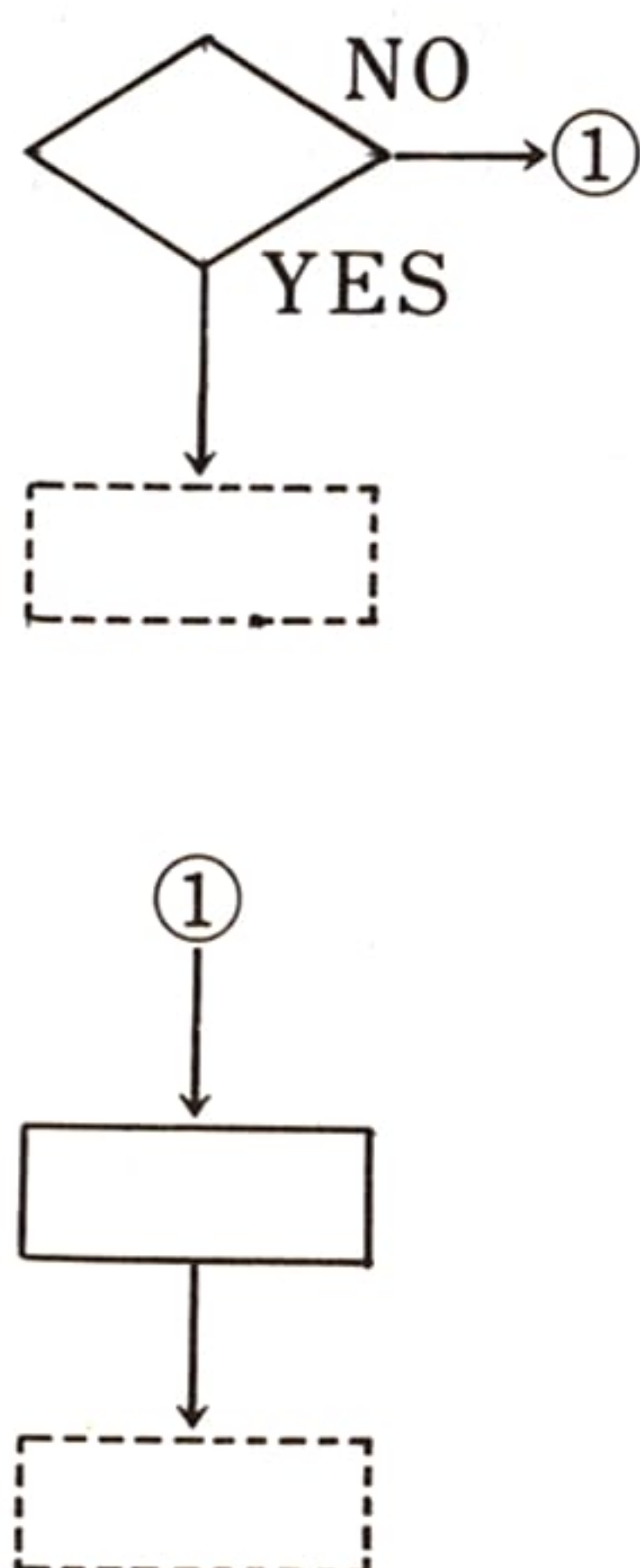
流れ図を書くときに使う主な記号は、つぎの6つです。

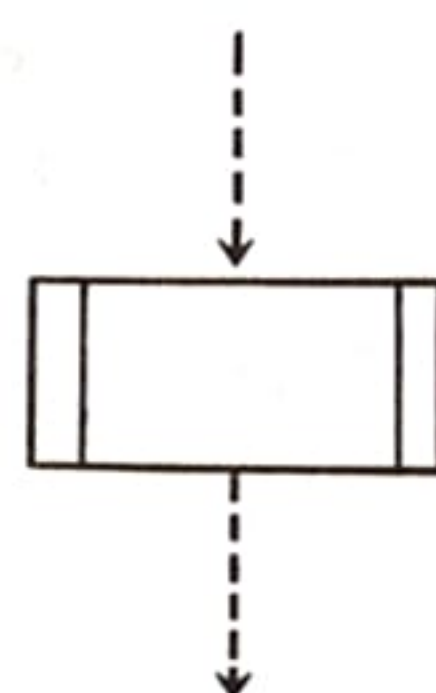
① ターミナル

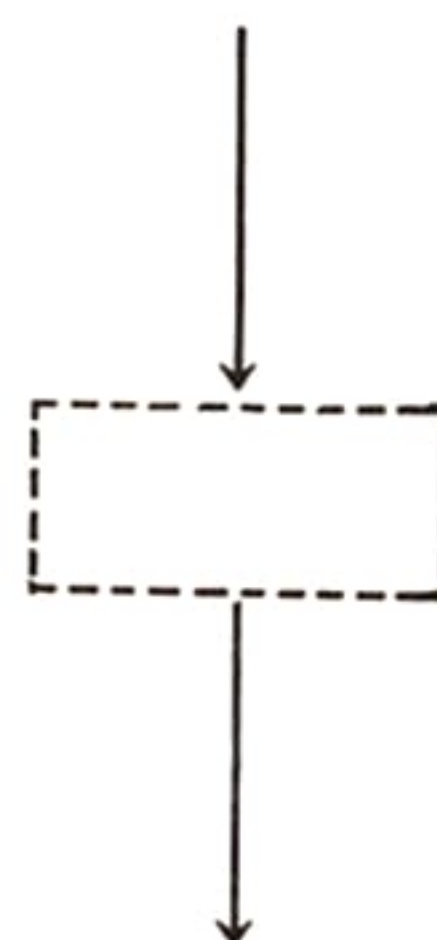
ターミナルという記号は、ひとつの流れ図のはじめと、おわりを表す記号です。

② 判　断

判断の記号は、ある質問に対して、その答えいかんで、つぎに実行する箇所を指し示すものです。たとえばCという数値が、5以下ならAという箇所にとび、6以上なら、つぎのBという箇所にいくとするなら、判断を表す記号のなかにC≦5と書いて、YESならAへ、NOならBにいくように線を引きます①。また②のように書くこともあります。

③ 各種処理

データの処理を表します。たとえば、Cの数値に1をたし、その値をYに代入するなどの記号です。

⑤ コネクタ

フローチャートの出口や入口を表します。◯のなかに数字を入れて、流れていく先を示します。

④ 定義済み処理

別の流れ図で定められた処理手順を表します。だいたいサブルーチンを表すことが多いようです。サブルーチンとは繰り返し使われる手順をひとまとめにして、本筋のプログラムから切り離した副プログラムのことです。

⑥ 流れ線

これまでおはなしした記号を結ぶ矢印の記号で、手順の流れていく方向を示します。

NEW

NEWは、メモリのなかをきれいにするコマンド(命令)

パソコンは、キーボードから入力されたプログラムや、カセットテープから入力されたプログラムを、メモリのなかに取り込んで記憶し、そして仕事をします。これを、プログラム内蔵方式と呼んでいます。

メモリのなかに記憶されたプログラムは、電源を切れば消えてしまいます。だからといって、新しくプログラムを入力しようとするたびに、電源を切っていたのでは、パソコンがたまりません。故障の原因にもなります。

そこで、NEWというコマンド（命令）を使います。NEWとは、メモリのなかに記憶されている内容を、すべて消す命令です。

NEWとキーボードのキーを叩いて、RETURNキーを押すと、メモリのなかは、きれいに掃除されているはずです。

消えているかどうかを確認するには、LIST をとってみればわかります。　LISTと入力し、RETURNキーを押すと、メモリのなかが、きれいに掃除されていれば、

```
list
OK
```

という表示になります。

ところで、まえに入力したプログラムが、メモリのなかに残っているところに、新しくプログラムを入力したら、どうなるでしょうか。

```
10 READ A,B
15 S=A*B/2
20 PRINT S
30 DATA 15,8
40 DATA 26,17
50 DATA 33,10
60 DATA 9,5
70 END
```

まえに入力したプログラム

```
10 INPUT R
20 S=3.14*R*R
30 PRINT S
40 END
```

新しく入力したプログラム

```
10 INPUT R
15 S=A*B/2
20 S=3.14*R*R
30 PRINT S
40 END
50 DATA 33,10
60 DATA 9,5
70 END
```

まえに入力したプログラムと新しいプログラムとが、ごっちゃになったプログラム

行番号がおなじ場合は、あとから入力したものにかわりますが、行番号が違っていると、そのまま残ってしまい、新しいプログラムと、まえのプログラムが、入り混ったものになってしまいます。

もちろん、これは極端な例ですが、このようなことが起こらないよう、ふつう、新しくプログラムをパソコンに入力するときは、まずNEWを入力して、メモリのなかをきれいにし、それからプログラムを入力する、といった方法をとっています。

二重添え字つき変数とDIM

二重添え字つき変数は、２つの添え字をもっている

添え字つき変数は、117頁でおはなししたように、カッコのなかにひとつの添え字をもっているものでした。

A（5）　C3（8）　S（I）　T（X）

添え字つき変数は、添え字がひとつ

これに対して二重添え字つき変数は、カッコのなかの添え字が、コンマで区切って２つあります。

A（3，5）　S（M，N）　X（I，I）

二重添え字つき変数は、添え字がふたつ

この二重添え字つき変数も、添え字つき変数とおなじように、カッコのなかの添え字は数値でも変数でも、また数式でもかまいませんが、－1とか－6とかいった負の整数ではなくて、5とか8とかいった正の整数でなければなりません。

さて、添え字つき変数は、DIM　A（7）とすると、メモリのなかに、A（0）からA（7）までの記憶場所を確保しました。

では、二重添え字つき変数は、どのようにメモリのなかに記憶場所を確保するのでしょうか。

もちろん、二重添え字つき変数の場合も、その頭にDIMをつけなければなりません。

DIM　A（3，5）

このようにすると、つぎのようにメモリのなかに、Aという変数の記憶場所が確保されます。

	0列	1列	2列	3列	4列	5列
0行	A(0, 0)	A(0, 1)	A(0, 2)	A(0, 3)	A(0, 4)	A(0, 5)
1行	A(1, 0)	A(1, 1)	A(1, 2)	A(1, 3)	A(1, 4)	A(1, 5)
2行	A(2, 0)	A(2, 1)	A(2, 2)	A(2, 3)	A(2, 4)	A(2, 5)
3行	A(3, 0)	A(3, 1)	A(3, 2)	A(3, 3)	A(3, 4)	A(3, 5)

つまり、DIM　A（３，５）とすると、Aは３つの行と、５つの列をもっているということを、パソコンに知らせます。行は、縦にとられていきます。列は横にとられます。

パソコンの行や列は、０行、０列からはじまるので、うえの図のように、０行や０列も同時に確保されます。実際には、０行や０列は、確保されてもあまり使われません。

このように、DIM　A（３，５）とすると、合計24個の記憶場所が、メモリのなかに確保されるのです。

では、この二重添え字つき変数は、どういったときに使うのでしょうか。

二重添え字つき変数は、私たちが日常、表で表しているものを処理

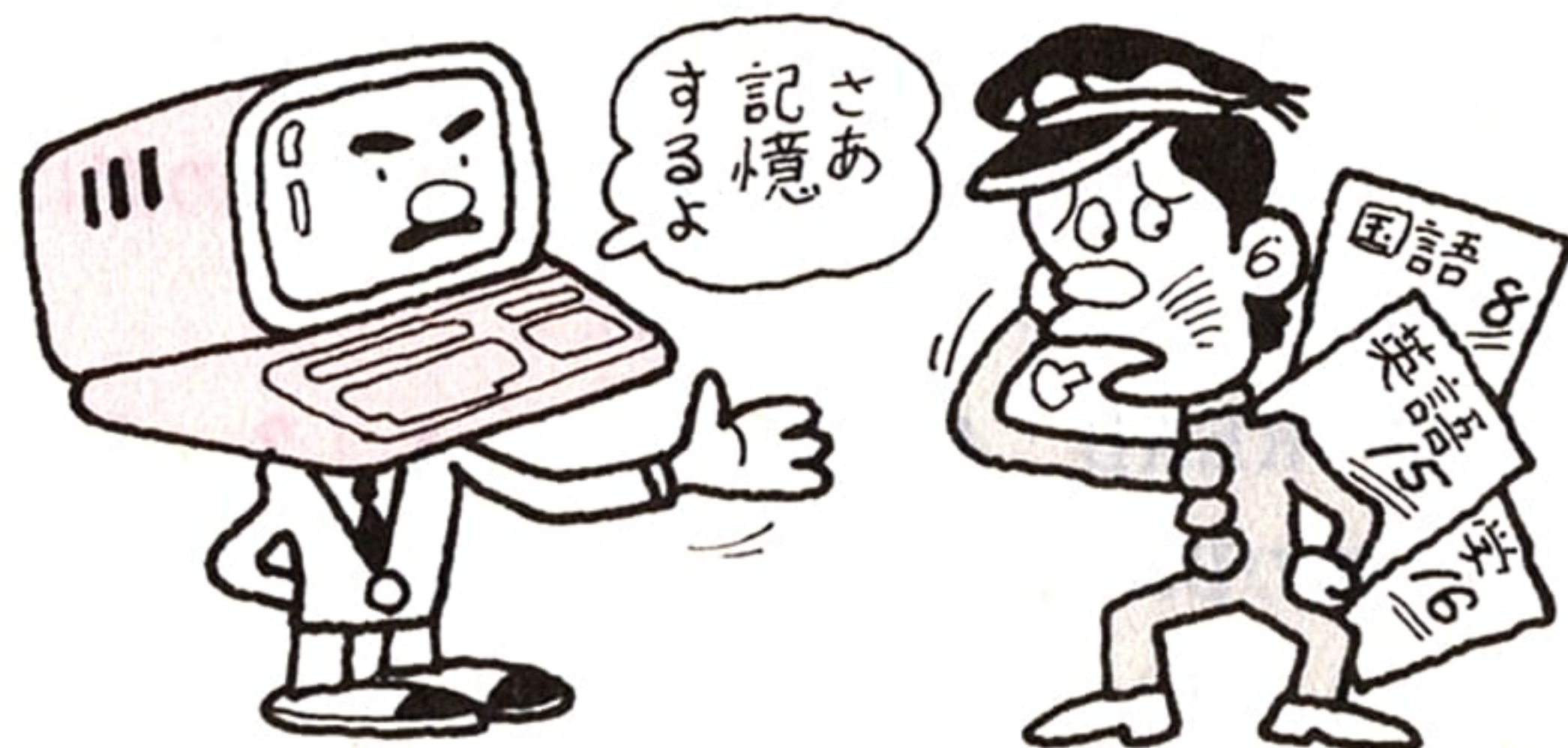
するのに役立ちます。

たとえば、ここに5人の学生がいて、その5人の学生の英語、数学、国語の3科目の成績を表すときなど、私たちはつぎのように表にして、表します。

		1列	2列	3列	4列	5列
	学生名 / 学科	森	橋　本	鈴　木	佐　藤	福　田
1行	英　語	85	90	75	95	80
2行	数　学	73	95	62	90	82
3行	国　語	65	83	80	93	98

この表に英語を1行、数学を2行、国語を3行とし、学生の名前の欄を順番に、1列、2列、3列としていくと、さきのメモリのなかに確保された記憶場所の図に、よく似ていることに気づくでしょう。つまり、二重添え字つき変数は、パソコンのメモリのなかに、このような表を作って処理するといえます。

では、この成績表を例にして、二重添え字つき変数の働きについてみてみましょう。

この成績表の場合も、学科が3科目で、学生数は5名ですから、この成績を全部入れるために確保する記憶場所は、3行と5列というこ

とになります。これを二重添え字つき変数で表すと

DIM　A（3，5）

となります。

このDIM　A（3，5）を使って、メモリのなかに記憶場所を確保して、各学生の成績をその記憶場所に入れるプログラムは、つぎのようになります。

```
10 DIM A(3,5)
20 FOR I=1 TO 5
30 READ A(1,I),A(2,I),A(3,I)
40 NEXT I
50 DATA 85,73,65,90,95,83,75,62
60 DATA 80,95,90,93,80,82,98
70 END
```

ほんとうは、二重FOR～NEXTループを使って、確保した記憶場所に、各学生の成績を入れるところなのですが、二重FOR～NEXTの働きは、初めての人には理解がむずかしいので、FOR～NEXTをひとつにしました。

このプログラムをパソコンのなかに入力して、RUNを入力すると、行番号10のDIM　A（3，5）を実行して、一番最初の図のように、メモリのなかに記憶場所が確保されます。

行番号20と40は、FOR～NEXTループで、Iが1から5になるまで処理をしなさいということです。なにを処理しなさいということなのかというと、行番号20のFORと行番号40の間にはさまれている行番号30のREADをということです。

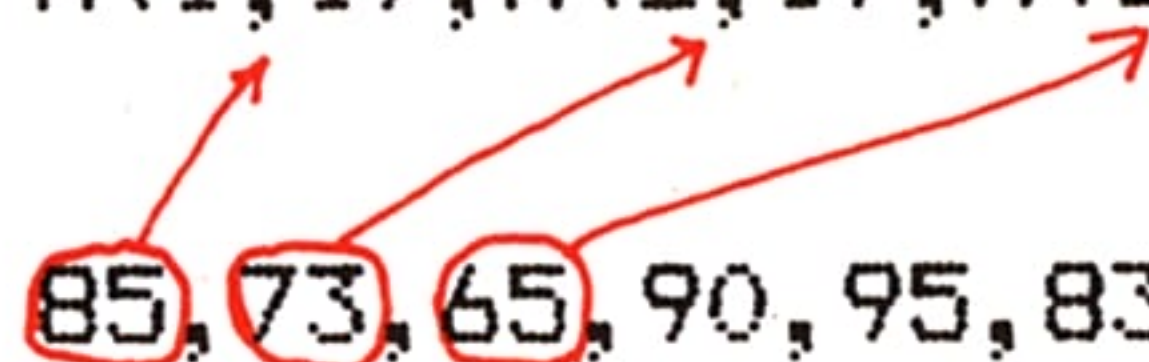
行番号30のREAD　A（1，I）、A（2，I）、A（3，I）は、行番号50と60のDATAに示されている数値を、A（1，I）、A（2，I）、A（3，I）に読みとりなさいということです。FOR～NEXT

は、このREADに働きかけて、5回データを読みとらせるのです。

では、この様子を順を追ってみていきましょう。

まず、FOR　I＝1　TO　5は、Iが1からですから、Iは1になり、このIの中身の1は、行番号30のすべてのIに送り込まれます。

```
20 FOR I=1

30 READ A(1,I),A(2,I),A(3,I)
```

したがって、行番号30は、つぎのようになります。

30　READ　A（1，1）、A（2，1）、A（3，1）

READは、DATAに示されている数値を読みとりなさいということです。そこで、第1回目は、つぎのように読みとります。

```
30 READ A(1,I),A(2,I),A(3,I)

50 DATA 85,73,65,90,95,83,75,62
```

DATAに示されている数値は、最初から順番に読みとられます。したがって、点数の並べ方には、注意しなければなりません。

READによって、A（1，1）に読みとられた数値85は、メモリの

なかに確保されたA（1，1）にしまわれて、記憶されます。A（2，1）、A（3，1）に読みとられた数値、73、65もおなじように、メモリのなかに確保されたA（2，1）、A（3，1）のなかにしまわれて記憶されます。

数値を読みとると、パソコンは、行番号40のNEXT　Iを実行します。NEXTは、Iの中身に1をたす働きをするところです。いつも1をたすとはかぎりません。このことについては、166頁をみてください。

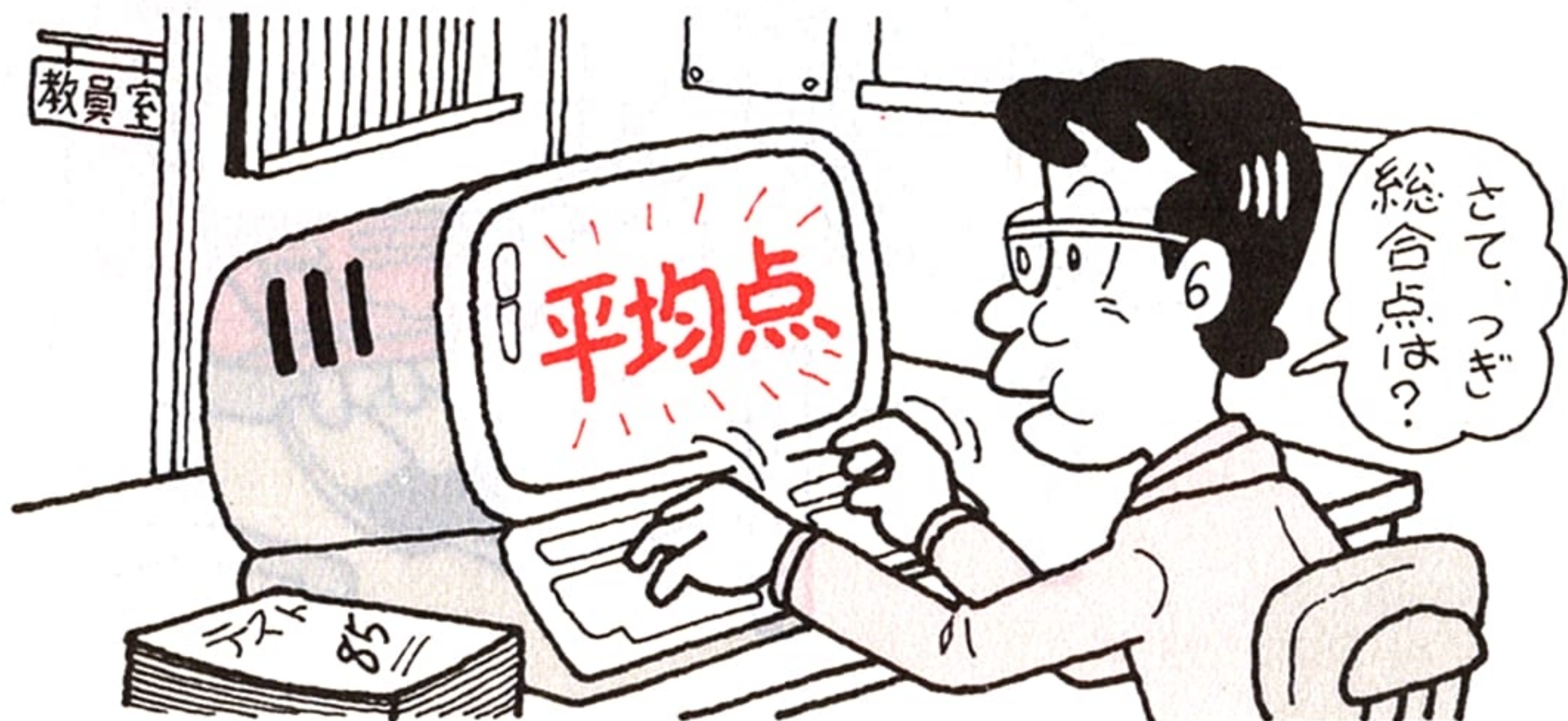
Iの中身は1でしたから、それに1をたすと、2になります。この2は、行番号20のIに送り込まれます。したがって、こんどは、I＝2、つまりIの中身は2になります。このIの中身2は、また、行番号30のすべてのIに送り込まれます。こんどは、行番号30がどうなるか、もうわかるでしょう。

30　READ　A（1，2）、A（2，2）、A（3，2）

になります。そして、DATAに示されている数値を読み込みます。

```
30 READ A(1,I),A(2,I),A(3,I)

50 DATA 85,73,65,90,95,83,75,62
```

A（1，2）に読みとられた90、A（2，2）に読みとられた95、A（3，2）に読みとられた83は、メモリのなかに確保されたA（1，

2）、A（2，3）のなかにしまわれて、記憶されます。

このことをIが5になるまでくり返すと、メモリのなかに確保された記憶場所には、つぎのように成績が記憶されることになります。

A(1, 1)	A(1, 2)	A(1, 3)	A(1, 4)	A(1, 5)
85	90	75	95	80
A(2, 1)	A(2, 2)	A(2, 3)	A(2, 4)	A(2, 5)
73	95	62	90	82
A(3, 1)	A(3, 2)	A(3, 3)	A(3, 4)	A(3, 5)
65	83	80	93	98

パソコンには、英語、数学、国語といった科目名や、森、橋本などといった学生の名前などは、まったく関係ありません。A（1，1）に85、A（2，1）に73、A（3，1）に65といったことを読み込んで記憶しているにすぎません。

A（1，1）の85は、森の英語の成績、A（2，1）の73は、森の数学の成績、A（3，1）の65は、森の国語の成績といったことは、私たちが記憶しておかなければならない事柄なのです。

では、このようにメモリのなかに記憶された点数を、パソコンにどう処理させたらよいでしょうか。このどう処理させたらよいかを考えるのは、私たちであり、その処理の仕方を考えて、その手順をまとめたものが、プログラムなのです。

このように記憶させた成績を取りだして、各科目の平均点を求めることもできます。また、各学生の合計点を求めることもできます。さらに、全体の総合点を求めることもできます。

このように1度メモリのなかに記憶された成績は、電源を切るか、

メモリのなかを掃除するＮＥＷコマンドを入力しない限り消えませんから、何回でも取りだして、利用することができるのです。

　では、ここでは、英語、数学、国語といった各科目の平均点を求めてみます。

```
10 DIM A(3,5)
20 FOR I=1 TO 5
30 READ A(1,I),A(2,I),A(3,I)
40 NEXT I
50 A1=(A(1,1)+A(1,2)+A(1,3)+A(1,4)+A(1,5))/5
60 A2=(A(2,1)+A(2,2)+A(2,3)+A(2,4)+A(2,5))/5
70 A3=(A(3,1)+A(3,2)+A(3,3)+A(3,4)+A(3,5))/5
80 PRINT "ｴｲｺﾞﾉﾍｲｷﾝﾃﾝ";A1
90 PRINT "ｽｳｶﾞｸﾉﾍｲｷﾝﾃﾝ";A2
100 PRINT "ｺｸｺﾞﾉﾍｲｷﾝﾃﾝ";A3
110 END
120 DATA 85,73,65,90,95,83,75,62
130 DATA 80,95,95,93,80,82,98
run
ｴｲｺﾞﾉﾍｲｷﾝﾃﾝ 85
ｽｳｶﾞｸﾉﾍｲｷﾝﾃﾝ 81.4
ｺｸｺﾞﾉﾍｲｷﾝﾃﾝ 83.8
```

　DIM　A（３，５）で、メモリのなかに記憶場所を確保して記憶させたデータを使うときは、うえのプログラムの行番号50、60、70に示すように、その記憶させた場所を指定します。

　このようにするとパソコンは、その記憶場所からデータを取りだして、処理します。

STICK

STICKは矢印のキーなどが、どの方向に押されているかを調べる

STICK（スティック）は、矢印のキー（カーソルコントロールキー）やジョイスティックが、どの方向に押されているか調べます。

STICKのあとにカッコで囲んで0とすると、

STICK　（0）←――カーソルコントロールキーの場合

カーソルコントロールキーが、どの方向に押されているかを調べます。STICKのあとにカッコで、1あるいは2とすると、

STICK　（1）
STICK　（2）　←――ジョイスティックの場合

ジョイスティックが、どの方向に押されているかを調べます。

STICK（1）は、ジョイスティックをジョイスティック端子の1につないでいるときに使います。STICK（2）は、ジョイスティックをジョイスティック端子2につないでいるときに使います。

ここでは、カーソルコントロールキーの場合を取りあげて、STICKの働きを説明することにします。つまり、

STICK　（0）

としたときの働きです。ただし、STICKは関数ですから、STICK（0）だけでは使うことができません。必ずつぎのように、

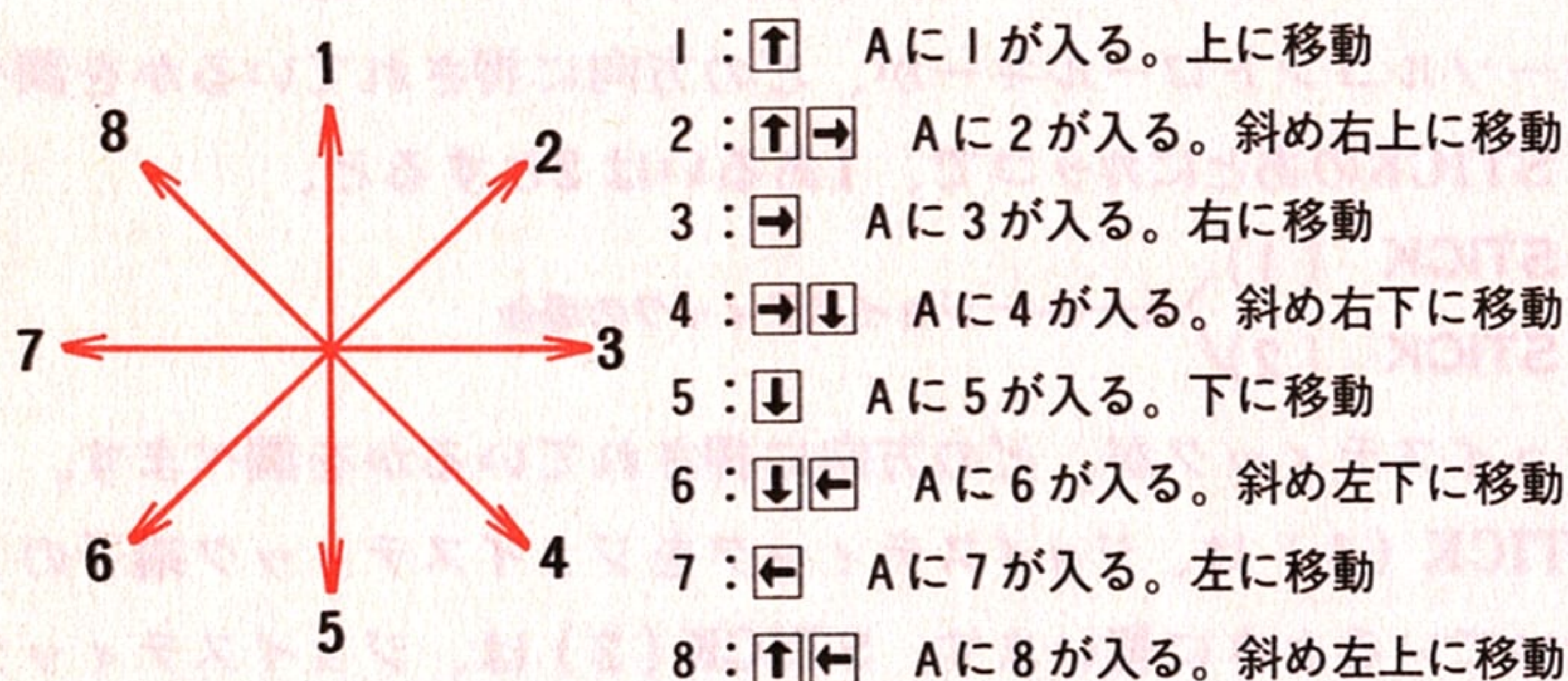
A＝STICK　（0）

ステートメントにしなければなりません。

さて、このようにして[↑]のキーを押すと、＝の左側にあるAには1の値が入ります。[→]のキーを押すと、Aには3の値が入ります。[←]の

キーを押すと、Aには7の値が入ります。⬇のキーを押すと、Aには5の値が入ります。

ところでカーソルコントロールキーには、斜め方向を示している矢印のキーがありません。斜め左上の場合は、⬅のキーと⬆のキーを押します。すると、Aには8の値が入ります。斜め右上の場合は、⬆のキーと➡のキーを押します。すると、Aには2の値が入ります。斜め左下の場合は、⬅のキーと⬇のキーを押します。すると、Aには6の値が入ります。斜め右下の場合は、⬇のキーと➡のキーを押します。すると、Aには4の値が入ります。

このことを図に示すと、つぎのようになります。

このようにSTICKは、押されたキーによって1から8までの値を取り出す働きです。したがって、STICKだけではキャラクタなどを動かすことができません。そこでSTICKを使って、どのようにすればキャラクタなどを動かすことができるのか、説明しておくことにしましょう。

つぎのプログラムの行番号50にSTICK(0)が使われています。このプログラムを実行させると、画面の真中あたりに●が表示されます。表示された●は、⬆のキーを押すと上に移動します。⬇のキーを押すと●は下に、⬅のキーを押すと●は左に、➡のキーを押すと●は右に

移動します。なお、このプログラムは●の移動の制限をしていませんから、画面の上下、左右いっぱいまで、●を移動させないようにしてください。

```
10 CLS:KEY OFF
20 X=12:Y=10
30 LOCATE X,Y
40 PRINT "●"
50 A=STICK(0)
60 LOCATE X,Y
70 PRINT " "
80 IF A=1 THEN Y=Y-1
90 IF A=3 THEN X=X+1
100 IF A=5 THEN Y=Y+1
110 IF A=7 THEN X=X-1
120 GOTO 30
```

さて、●がはじめに画面の真中あたりに表示されるのは、行番号20でXに12、Yに10を代入しているからです。この値が、行番号30のLOCATE　X, Yに送られてきて●の表示位置を指定し、その位置にPRINTで●が表示されます。

行番号50のA＝STICK（0）で、カーソルコントロールキーのどのキーが押されたかを判定します。キーが押されないときは、STICK（0）で、0の値が取り出されてAに入ります。0の値のときは移動はしません。

行番号60のLOCATE　X, Yと行番号70のPRINT〝　〞は、行番号30のLOCATE　X, Yで指定して、行番号40のPRINTで表示した●を消します。行番号60のLOCATE　X, YのXとYには、行番号30の

LOCATE X, YのXとYの値とおなじ値が送られてきます。したがって、表示した●の位置に空白が表示されますから、●は消えることになります。画面の真中に表示した●が点滅しているのは、●を表示したり消したりしているためです。

では、⬆のキーを押します。するとSTICK(0)で1がAに入ります。したがって、行番号80のIF A=1 (Aの内容が1と等しいこと)という条件を満たしますから、THEN Y=Y−1を実行します。Yには行番号20で、10が代入されていますから、

10 − 1=9

が行われて、Yの値は9になります。したがって、行番号30のLOCATE X, YのYの値が、

```
30 LOCATE 12, 9
```

となって、ひとつ上の位置に●が表示されます。⬆のキーを押しつづけると、9−1=8、8−1=7、7−1=6が行われて、●がどんどん上に向かって移動していきます。

➡のキーを押すと、Aには3が入ります。したがって、行番号90のIF A=3(Aの内容が3と等しいこと)という条件を満たして、THEN X=X+1を実行します。Xには行番号20で、12が代入されていますから、

12+1=13

が行われて、Xの値が13になります。したがって、行番号30のLOCATE X, YのXの値が、

```
30 LOCATE 13, 10
```

となって、ひとつ右の位置に●が表示されます。➡のキーを押しつづけると、13+1=14、14+1=15、15+1=16が行われて、●が右に向かってどんどん移動していきます。

⬇のキーを押したときは、Aに5が入ります。したがって、行番号100のIF A=5の条件を満たして、THEN Y=Y+1を実行します。

この場合、[↑]のキーを押して●を画面の上に移動していると、Yの値は、●が表示されている位置の値となります。ここでは、●が、

LOCATE 12, 3

の位置にあるものとしましょう。とすると、Yの値は3ですから、

3+1=4

が行われて、Yの値は4になります。したがって、行番号30のLOCATE X, YのYの値が、

30 LOCATE 12, 4

となって、ひとつ下の位置に●が表示されます。[↓]のキーを押しつづけると、4+1=5、5+1=6、6+1=7が行われて、●は下に向かってどんどん移動していきます。

[←]のキーを押すと、Aには7が入りますから、行番号110のIF A=7の条件を満たして、THEN X=X-1を実行します。この場合も、[→]のキーを押して、●が画面の右に移動しているとすると、Xの値は●が表示されている位置の値となります。ここでは●が、

LOCATE 20, 10

の位置にあるものとします。とすると、Xの値は20ですから、

20-1=19

が行われて、Xの値は19になります。したがって、行番号30のLOCATE X, YのXの値は、

30 LOCATE 19, 10

となりますから、ひとつ左の位置に●が表示されることになります。[←]のキーを押しつづけると19-1=18、18-1=17、17-1=16が行われて、●が左に向かってどんどん移動することになります。

これまで、説明したことが繰り返し行われるのは、行番号120のGOTO 30で、実行を行番号30に戻しているからです。このGOTO 30がないと、●を移動させることはできません。

RENUM

RENUMは、行番号をつけなおす

プログラムを作っていくと、ステートメントを追加したりなどしてつぎに示すように、行番号が不揃いになります。

```
10 CLS:KEY OFF
15 WIDTH 32
18 GY=11
20 FOR I=1 TO 10
30 BY=INT(RND(1)*22)+1
40 FOR J=1 TO 27
45 COLOR 8,10
47 LOCATE J,BY
50 PRINT " ●"
56 FOR T=1 TO 50:NEXT T
65 NEXT J
70 LOCATE 28,BY
90 PRINT " "
110 NEXT I
120 END
```

このようなとき行番号を10おきに揃えるには、ダイレクトモードで、

renum ← RETURNキーを叩く

と入力して、RETURNキーを叩きます。このようにすると、メモリのなかでは、行番号10からはじまって、10おきに行番号が整理されてつけなおされます。

これを確めるには、メモリのなかのプログラムを画面に表示させなければなりません。メモリのなかのプログラムを画面に表示させるコマンドは、LISTです。

LIST ←――RETURNキーを叩く

と入力してRETURNキーを叩きます。この場合、LISTと入力してRETURNキーを叩いてもかまいませんし、LISTはファンクションキーの[F 4]のキーですから、[F 4]キーを叩いてRETURNキーを叩いてもかまいません。すると、行番号がつけなおされたプログラムが、つぎのように画面に表示されます。

```
10 CLS:KEY OFF
20 WIDTH 32
30 GY=11
40 FOR I=1 TO 10
50 BY=INT(RND(1)*22)+1
60 FOR J=1 TO 27
70 COLOR 8,10
80 LOCATE J,BY
90 PRINT " ●"
100 FOR T=1 TO 50:NEXT T
110 NEXT J
120 LOCATE 28,BY
130 PRINT "  "
140 NEXT I
150 END
```

プリンタへの出力

LLIST、LPRINTはプリンタへ出力させる

パソコンへプログラムを入力すると、そのプログラムは、パソコンのメモリのなかへ記憶されます。

パソコンのメモリのなかへしまわれたプログラムを、ディスプレイの画面のうえに表示させる命令は、LISTです。

```
list
```
← LISTと入れて、RETURNキーを押します

キーボードのキーをたたいてLISTと入力するか、PC－8001の場合はf･4キーを押してRETURNキーを押すと、メモリのなかにしまわれているプログラムが、全部、ディスプレイの画面のうえに表示されます。

```
10 READ A,B
20 S=A+B
30 PRINT S
40 GOTO 10
50 DATA 53,24,183,149,123,165,179,198
60 END
```

プリンタへの出力は、LIST の頭に、Lをひとつよけいにつけた、LLISTを使います。

```
llist
```
← LLISTと入力して、RETURNキーを押します

これをキーボードのキーをたたいて入力し、RETURN キーを押すと、プログラムは全部、プリンタから印字されて出力されます。

ただし、LLISTは、プログラムをプリンタから出力させるだけで、実行結果は、LLISTでは出力されません。

実行結果をプリンタから印字させて出力させるためには、

```
LPRINT
```
←実行結果をプリンタから出力させます

を使います。

このLPRINTは、PRINTとおなじように、あらかじめプログラムのなかに入れておかなければなりません。

```
10 READ A,B
20 S=A+B
30 LPRINT S
40 GOTO 10
50 DATA 53,24,183,149,123,165,179,198
60 END
run
77
332
288
377
```

実行結果を、プリンタから出力させたいときは、あらかじめ、LPRINTをプログラムのなかに入れておきます

行番号30のように、LPRINT　Sとプログラムのなかに入れておくと、S＝A＋Bの実行結果が、ディスプレイに表示されるのと同じように、プリンタから印字されて出力されます。

PRINT

PRINTは実行結果などをディスプレイに表示させる

PRINTはプログラムのなかに、もっとも多くみられることばです。

私たちが、パソコンに仕事をやらせても、その結果を知ることができなければ、どうしようもありません。そこでPRINTということばがあるのです。

PRINTは、「ディスプレイの画面に、仕事の結果を表示しなさい」という、パソコンに対する命令で、パソコンは、PRINTということばに出合うと、その仕事の結果を、ディスプレイに表示して、私たちに知らせてくれます。

もちろん、PRINTは、このようなことばかりでなく、PRINTのあとにつづく数値、クォーテーションマークで囲まれた文字列（ストリング）や記号なども表示します。

```
10 PRINT "ｺﾝｹﾞﾂﾉｼｭｳﾆｭｳﾊ";
20 INPUT A
30 PRINT "ｺﾝｹﾞﾂﾉｼｼｭﾂﾊ";
40 INPUT B
50 S=A-B
60 PRINT "ｺﾝｹﾞﾂﾉｶｹｲ:ｻｼﾋｷ";S
70 END
```

うえのプログラムでは、行番号10、30、60にPRINTが使われてい

ます。パソコンの仕事の結果の表示は、行番号60の PRINT　S です。PRINT と S の間にある〝コンゲツノカケイ：サシヒキ〟という文字列は、S を説明する文です。

行番号10の PRINT 〝コンゲツノシュウニュウハ〟という文字列は、つぎの行番号20の A に、どんなデータを入れるのか、案内をする文です。この文は、プロンプト文またはガイド文と呼んでいます。行番号を30の〝コンゲツノシシュツハ〟もおなじで、これは行番号40の B に、どんなデータを入れるのか、案内をしています。このようにガイド文を用いると、そこにどんなデータを入力したらよいのかかわるので、誰もがデータを間違えないで入力できるようになります。

では、どんな具合になるのか、このプログラムを実行しておはなししましょう。RUN を入力すると、パソコンは行番号10を実行し、つづいて行番号20を実行して、つぎのようにディスプレイの画面のうえに表示します。

```
run
コンゲツノシュウニュウハ?
```

?マークがコンゲツノシュウニュウハのあとにきているのは、10 PRINT 〝コンゲツノシュウニュウハ〟のあとに「；」セミコロンをつけているからです。このセミコロンをとると、つぎのような表示になります。

```
run
コンゲツノシュウニュウハ
?
```

どのような形をとっても、実行結果はおなじなのでかまいませんが、いろいろな表示方法をおぼえていくことも、大切なことです。

さて、はなしをもとに戻します。

コンゲツノシュウニュウハ？の問いに対して、今月の収入を25万円として、250000を入力し、そして RETURNキーを押すと、パソコン

は行番号30、40を実行して、今月の支出について尋ねてきます。

```
run
コンゲツノシュウニュウハ？　250000
コンゲツノシシュツハ？　245000
```

250000を入力して、RETURNキーを押すと、つぎの文を表示してくる

245000を入力してRETURNキーを押す

そこで、今月の支出を24万5千円として、245000を入力して、RETURNキーを押すと、パソコンは、行番号50のS＝A－Bの計算式にもとづいて、計算をし、その計算結果を＝の左側のSに入れ、そして、行番号60のSに送ります。

行番号60の PRINT が実行されると、つぎのように、まず文字が表示され、つづいて、パソコンが仕事をした結果である、Sの中身が表示されます。

```
run
コンゲツノシュウニュウハ? 250000
コンゲツノシシュツハ? 245000
コンゲツノカケイ:サシヒキ 5000
```

パソコンが働いた結果

このように PRINT は、パソコンが働いた結果や、また、数値や、クォーテーションマークで囲まれた文字列（ストリング）などを、ディスプレイの画面のうえに表示する働きをするものです。

▶PRINTのあとに何も置かないと

これまで説明してきたPRINTのあとには、変数を置いたり、クォーテーションマークで囲んだ文字列（ストリング）を置きました。そして、パソコンが、そのPRINTを実行すると、変数に記憶されている値や文字列を画面に表示しました。

ではPRINTのあとには、必ず変数や文字列などを置かなくてはならないのかというと、そうではありません。つぎのように行番号をつけてPRINTだけを置くこともできます。

```
20 PRINT
```

さて、このようにPRINTのあとに何も置かないとどうなるのでしょうか。このようにPRINTのあとに何も置かないと、1行分の空白を作ります。2行分の空白を作りたいときは、

```
40 PRINT
50 PRINT
```

とします。

つぎに、PRINTだけを置いたプログラムを示します。行番号20にはPRINTをひとつだけ置いていますから、プログラムの下に示すように1行分の空白を作ります。行番号40と50には、PRINTだけを2つつづけて置いていますから、プログラムの下に示すように、2行分の空白を作ることになります。

```
10 PRINT "キョウノニュキン"
20 PRINT
30 PRINT "クモリ ノチ ハレ"
40 PRINT
50 PRINT
60 PRINT "アス ノ テンキ ハ フメイ"

Ok
run
キョウノニュキン

クモリ ノチ ハレ



アス ノ テンキ ハ フメイ
```

プログラムの修正とLIST

プログラムの間違いを修正、修正したらLISTで確認

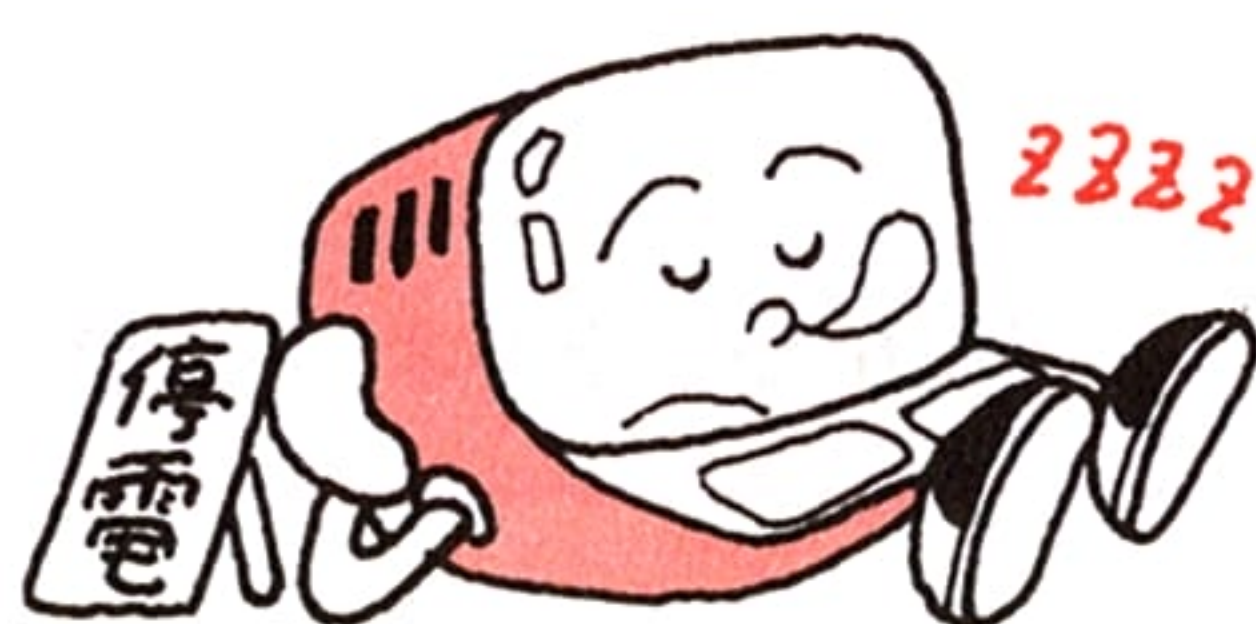
パソコンのなかにプログラムを入力するには、キーボードのキーをひとつひとつ叩いて行います。このプログラムの入力は、大変に手間のかかるものです。

特に日本のパソコンは、小さなキーのうえに、記号や数字、アルファベットやカナなど、2つから4つ書かれています。したがって、慣れるまでは、必ずタイプミスが生じます。

```
10 A=2
20 A=3
30 C=A+B
40 PRYNT C
50 END
```

カーソルをここまでもってきて、Bのキーをたたきます。そしてRETURNキーを押します。

カーソルをここまでもってきて、Iのキーをたたきます。そしてRETURNキーを押します

うえのプログラムのように、BであるところがAになっていたり、Iであるところが、Yになっていたときは、矢印のキーをたたいて、カーソルを修正する字のうえまでもってきて、BやIを入力して、RETURNキーを押します。RETURN キーを押さないと、文字の訂正は行なわれませんので、注意してください。

```
10 S=RND(1)
20 PRINT S
30 GOTO 10
40 END
50 REM
10 s=int(6*rnd(1))+1
```

行番号10の1行を全部修正するときは、このように行番号10として、プログラムの最後に入力してからRETURNキーを押すと、修正されます

このプログラムのように、行番号10の1行全部を修正したいときは、プログラムのおわりのほうに、行番号10としてステートメントを入力し、RETUNRキーを押しておくと、行番号10の1行全部が修正されます。

この場合とおなじ働きで、つぎのようにおなじ行番号が2つあると、まえのほうのステートメントは無効になり、あとのほうのステートメントがメモリのなかに記憶されます。

```
10 A=3
20 B=4
30 c=a*b
30 D=A/B
40 PRINT C,D
50 END
```

おなじ行番号が2つあると、まえのほうのステートメントがメモリのなかに記憶されません

あとから入力したステートメントがメモリのなかに記憶されます

ある行番号と、ある行番号との間に追加したいステートメントがあるとき、たとえば、つぎのプログラムに、D＝2、E＝(A＋B)＊Dを追加したいときは、

```
10 A=3
20 B=5
30 C=A+B
40 PRINT C,E
50 END
```

行番号35のEの中身を表示させるために、[,][E]も追加します。

```
25 D=2
35 E=(A+B)*D
```

追加したいステートメント

行番号20と30、行番号30と40の間の番号をとって、行番号として、プログラムのあとのほうに入力して、RETURN キーを押しておけば、新しいステートメントが追加されます。

```
10 A=13
20 B=15
30 C=5
40 D=(A+B)/C
50 E=(B-A)*C
60 PRINT D
70 PRINT E
80 END
40
60
```

行番号40を削除
行番号60を削除

1行全部を削除したいときは、削除したい行番号を入力して、RETURNキーを押すと、その行全部が削除されます。うえのプログラム

の場合、行番号40と60を削除するため、40と60を入力したものです。

以上がプログラムの修正の仕方です。

ところで、これまで行った修正が、正しく修正されているかどうか確認するには、どうしたらよいでしょうか。

このようなときに使うのが、LISTです。

LISTは、パソコンのメモリのなかに記憶されているプログラムを、ディスプレイの画面のうえに表示させる命令（コマンド）です。

LISTは[L][I][S][T]とキーボードから入力して、RETURNキーを押してもよいし、PC－8001では、ファンクションキーの[f･4]キーを押しても、おなじです。つぎのように

list（行番号の指定をしない）←プログラムを全部表示します

行番号を指定しないでLISTすると、全部のプログラムが、ディスプレイの画面のうえに表示されます。

list 30←行番号ひとつを指定すると、その行番号だけが表示されます

このようにLISTのあとに、ひとつの行番号を指定すると、その行番号だけのプログラムが、ディスプレイの画面のうえに表示されます。

list 80-90←このように指定すると行番号80から90までが表示されます

このように、LISTのあとに、ある行番号からある行番号までを指定すると、指定された行番号から行番号までのプログラムが、ディスプレイの画面のうえに表示されます。

list -50←このように指定すると、最初の行番号から行番号50までが表示されます

このようにLISTのあとに「—」を入力し、そのあとに行番号を指定すると、一番最初の行番号から、指定された行番号までのプログラムが、ディスプレイの画面のうえに表示されます。

list 30-←このように指定すると、行番号30から、おわりまでが表示されます

このようにLISTのあとに行番号を指定して、そのあとに「—」を入力すると、指定された行番号から、最後の行番号までのプログラムが、ディスプレイの画面のうえに表示されます。

変　数

変数は、数値をしまっておく記憶場所の名前

パソコンのメモリのなかは、一般に、小さな箱のような記憶場所の集まりと考えられています。

そして、BASICで書いたプログラムのなかの数値や数式は、A、B、Cといったアルファベットを使って、記憶させておくメモリのなかの記憶場所を指定します。

```
10 A=3
20 B=10
30 C=A+B
40 PRINT C
50 END
```

このプログラムをパソコンのなかに入力すると、パソコンは、メモリの記憶場所のなかから、A、B、Cのための3つの記憶場所を選びだして、つぎのように、数値や数式をしまって記憶します。

A	B	C			
3	10	A＋B			

このように、数値や数式をしまって記憶しておく、A、B、Cといった記憶場所の名前を、変数といいます。

変数の名前は、A、B、Cといったアルファベット26文字と、それに、A1、A2、A3といったようにアルファベット1文字に数字がひとつついたものです。

このなかから、自由に変数の名前を選んで指定すると、メモリのなかに、その名前の記憶場所ができて、その名前の記憶場所に入れる数値や数式を記憶させておくことができるといった仕組みになっています。

いままで、おはなししてきたのは、一般に変数といわれるものですが、このほかに、

添え字つき変数
ストリング変数
添え字つきストリング変数

といったものがあります（それぞれの項を参照してください）。

これらの変数も、一般に変数といわれるものとおなじで、メモリのなかに、数値や文字をしまって記憶しておく、記憶場所の名前です。

FOR～NEXT

FOR～NEXTは、繰り返して処理や計算をする

FOR～NEXTということばは、BASICでプログラムを書くとき、もっともよく使われることばです。ということは、それだけ便利なことばであるといえましょう。

このFOR～NEXTは、パソコンに、ひとつの仕事を、必要な回数だけ、繰り返し処理させたり、計算させるときに使います。

たとえば、1から5までの数を、ディスプレイに表示させたいとしましょう。もちろん、FOR～NEXTを使わないで、IF～THENとGOTOを使っても、1から5までの数をディスプレイに表示させることができます。つぎのようにです。

```
10 I=1
20 IF I>5 THEN 60
30 PRINT I
40 I=I+1
50 GOTO 20
60 END
```

このプログラムをパソコンに入力して、実行してみましょう。RUNを入力すると、パソコンは、I＝1の1を＝の左側のIに入れます。Iのなかに入った1は、行番号20のI＞5のIに送られます。行番号20のIF　I＞5　THEN　60は、もしIが5より大きければ60へ行けということで、I＞5で、Iの中身が5より大きいかどうか、比較判

断されます。Ｉの中身１は、５より小さいので、Ｉが５より大きいという条件を満していません。そこで、THEN　60の60は実行されないで、行番号30が実行されます。PRINT　Ｉの中身は行番号20のＩから送られてきた１です。PRINT　Ｉは、Ｉの中身を表示しなさいということですから、１が表示されます。

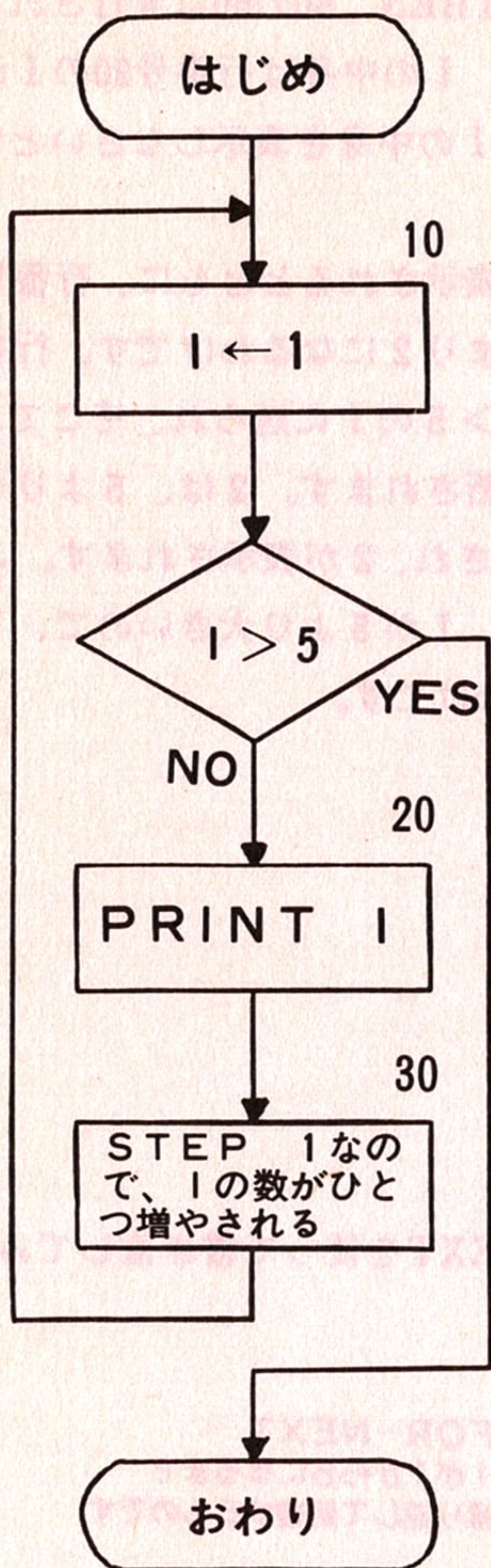
このＩの中身１は、ディスプレイに表示されるとともに、行番号40のＩに送られ、１プラスされます。つまり２になるわけです。行番号50のGOTO　20で、Ｉの中身２は、Ｉ＞５のＩに送られ、そこで、ふたたび５より大きいかどうか、比較判断されます。２は、５より小さいので、行番号30のPRINT　Ｉが実行され、２が表示されます。このように繰り返されて、Ｉが６になると、Ｉが５より大きいので、THEN　60が実行されて、おわるというわけです。

```
run
 1
 2
 3
 4
 5
```

では、このプログラムを、FOR～NEXTを使って書き直してみましょう。

```
10 FOR I=1 TO 5
20 PRINT I
30 NEXT I
40 END
```

くり返すところ

FOR～NEXT
Ｉが１から５になるまで
繰り返して処理するものです

FOR～NEXTの働き

実行結果、つまりディスプレイに表示されることはおなじことですがFOR～NEXTを使ったほうが、すっきりしたプログラムになっているとは、思わないでしょうか。それだけ、よけいなことばや、計算式を使わないですむということです。

ではまず、このプログラムを例にして、FOR～NEXTが、どのように繰り返し処理するのか、その働きをみてみましょう。

左の図をみてください。これがFOR～NEXTの働きを、図にしたものです。

行番号10の、FOR　I＝1　TO　5は、Iの中身が1からはじまって、5になるまで処理しなさいということです。

したがって、Iに1を入れることから、はじまります。

10と番号のついているワクの、I＝1がそれです。

つぎに、ひし形のワクがあります。これが、TO　5、つまり5までという働きをする部分です。パソコンが、Iの中身が5になったかどうかを知る場合には、Iの中身が5より大きいかどうか、比較判断して知る

という方法をとります。

20という番号がついているワクは、PRINT　I ですから、I の中身をディスプレイに表示します。

さて、30という番号のついているワクです。ここがNEXT　I です。NEXT　I は、つぎのI の中身をつくる部分です。この場合は、TO　5 のあとにSTEP　2 、STEP　3 などとなっていませんから、I の中身に1 をプラスします。1 をプラスしたI の中身は、行番号10のFOR　I のI に送り返されます。

では、実際にこのプログラムを入力して、実行してみましょう。RUNを入力すると、パソコンは、I に1 を入れて、その中身1 をI ＞ 5 のI に送り込みます。I の中身1 は、5 より小さいので、行番30のPRINT　I に送り込まれ、I の中身1 が表示されます。

このI の中身1 は、つぎに行番号40のNEXT　I に送られて、1 プラスされて2 になり、行番号10のFOR　I のI に送り返されます。I の中身2 は、また、ひし形のワクのなかのI ＞ 5 で比較判断されます。I の中身2 は5 よりも小さいので、行番号30のPRINT　I のI に送られ、2 が表示されます。そして、また行番号30のNEXT　I でプラス1 されて、こんどは3 になります。

これを　繰り返して、6 になったとき、ひし形のワクのなかのI ＞ 5

で比較され、6は5より大きいと判断されて、ひし形のワクの右端からでて、おわるということになります。実行結果は、つぎのようになります。

```
run
 1
 2
 3
 4
 5
```

これがFOR～NEXTの働きです。

FORとNEXTにはさまれている部分を含めて、これをFOR～NEXTループと呼んでいます。

また、まえにSTEP　2、STEP　3と書きました。

```
10 FOR I=1 TO 10 STEP 2
```

増分といいます

このSTEP　2　とか3を増分といいます。これは、2つずつ、あるいは3つずつ増やすということです。つまり、STEP　2　の場合は、NEXT　Iのところで、Iの中身に2がプラスされることになり

ます。さっきのプログラムの行番号10を、これと入れ変えて実行してみると、つぎのようになります。

```
run

1
3
5
7
9
```

1＋2＝3
3＋2＝5
5＋2＝7
7＋2＝9

STEP　3は、NEXT　Iのところで、3プラスされます。STEP　1のときは、STEP　1と書かなくてもよいことになっています。

FOR～NEXTのかたち

FOR　変数＝はじまりの値　TO　おわりの値　STEP　増やす数 減らす数

　　くり返す部分

NEXT　変数

正弦波を書くプログラム

```
10 SCREEN 2
20 W=2*3.14/159
30 FOR X=0 TO 159
40 PSET(X,40-40*SIN(X*W)),8
50 NEXT X
60 FOR T=1 TO 3000:NEXT T
```

COLOR

COLORは、色の指定を行う

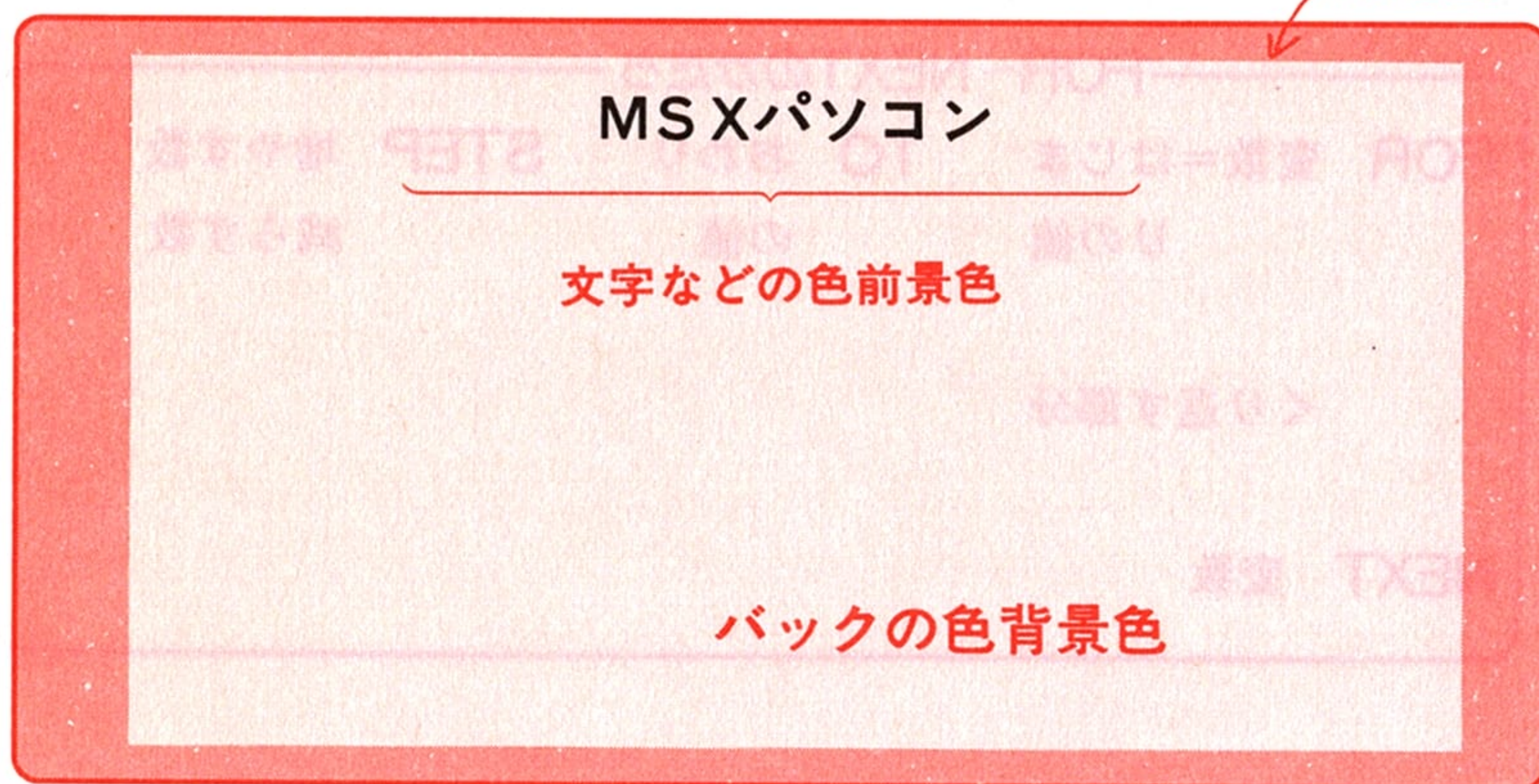
パソコンに電源を入れると、画面に表示される文字などの色は白、バックの色は青、画面の周辺の色は水色になっています。

画面に表示される文字などの色を、前景色といいます。バックの色は背景色、画面の周辺の色は周辺色といいます。

周辺色

MSXパソコン

文字などの色前景色

バックの色背景色

COLORは、パソコンに電源を入れたときに自動的につく色を、自分の好きな色に変えるために使います。COLORを使って自分の好きな色に変えることができるのは、前景色、背景色、周辺色の3か所ですから、COLORはつぎのように3か所の色指定ができます。

COLOR　〈前景色〉,〈背景色〉,〈周辺色〉

前景色、背景色、周辺色のところに指定することができる値は、0

から15までで、この値を、カラーコードといいます。0から15までのカラーコードは、つぎのように、それぞれの色に対応しています。

0	周辺色とおなじ色	6	暗い赤	12	暗い緑
1	黒	7	水色	13	紫
2	緑	8	赤	14	灰色
3	明るい緑	9	明るい赤	15	白
4	暗い青	10	黄色		
5	明るい青	11	明るい黄色		

では、COLORを使って色を指定してみることにします。

```
10 COLOR 10,12
20 PRINT "♥♥♥♥♥"
30 PRINT "♦♦♦♦♦"
40 END
```

前景色を黄色、背景色を暗い緑と指定して、ハートとダイヤを表示

上に示したプログラムを実行させると、画面に5個ずつ表示される♥と♦は黄色で表示されます。また、バックの色は暗い緑色になります。これは、行番号10のCOLORで前景色を10、背景色を12に指定しているからです。カラーコード10は黄色、カラーコード12は暗い緑です。なお、周辺色の指定は省略しているので、パソコンに電源を入れたときのままです。

ところで、うえのプログラムの場合、行番号20のPRINTで表示した♥も、行番号30のPRINTで表示した♦も、おなじ色で表示されました。行番号20のPRINTで表示した♥と、行番号30のPRINTで表示した♦を別の色で表示するには、行番号30のPRINTのまえにもCOLORで色の指定をしなければなりません。つまり、表示するものの色を変えるときは、つぎのプログラムに示すように、そのつど、COLORで指定しなければならないということです。

```
10 COLOR 10,12
20 PRINT "♥♥♥♥♥"   ← 黄色で表示される
30 FOR T=1 TO 3000:NEXT T   ← ウエイト・ループ
40 CLS   ← 画面の表示をすべて消す
50 COLOR 8
60 PRINT "◆◆◆◆◆"   ← 赤色で表示される
70 END
```

行番号10のCOLORの指定は、まえとおなじです。したがって、20のPRINTで♥が5個黄色で表示されます。バックの色も、まえとおなじく暗い緑です。

行番号30のFOR～NEXTは、行番号40の画面消去のCLSの実行を遅らせるためのものです。つまり、FORの変数Iの値が初期値1から最終値3000になるまで、FORとNEXTの間をいったりきたりしてCLSの実行が遅れますから、その間♥が表示されているのです。なぜ、CLSで画面を消去するのかは、あとで説明します。

行番号40のCLSで画面を消去すると、行番号50のCOLORで、行番号60のPRINTで表示する◆の色指定が行われます。この場合は、前景色がカラーコード8と指定されているだけで，背景色の指定、周辺色の指定が省略されていますから、◆の色だけが変更されます。カラーコード8は赤です。したがって、◆は赤で表示されます。

さて、行番号40のCLSで画面を消去している理由です。

行番号40のCLSを取って実行させてみるとわかりますが、♥を表示したまま行番号50のCOLOR　8を実行させると、行番号60のPRINTで表示される◆ばかりでなく、♥も赤に変わります。そこでCLS で画面を消去して、◆を表示させているのです。

PRINTで表示するものばかりでなく、バックの色や周辺色などを変えるときも、そのつど、COLORで色の指定を行えばよいのです。つ

ぎに、その例を示します。

```
10 COLOR 8,11
20 PRINT "♠♠♠♠♠"
30 FOR T=1 TO 3000:NEXT T
40 CLS
50 COLOR 4,15
60 PRINT "♣♣♣♣♣"
70 END
```

うえのプログラムの場合、行番号10のCOLORの指定は、前景色がカラーコードの8、背景色がカラーコード11です。したがって、行番号20で表示される♠は、赤で表示されます。バックは明るい黄色となります。行番号50のCOLORでは、前景色がカラーコード4、背景色がカラーコード15と指定されていますから、行番号60のPRINT で表示される♣は、暗い青で表示されます。バックは、白色に変わります。

```
10 COLOR 8,11,12
20 PRINT "♠♠♠♠♠"
30 FOR T=1 TO 3000:NEXT T
40 CLS
50 COLOR 4,15,9
60 PRINT "♣♣♣♣♣"
70 END
```

うえのプログラムでは、周辺色を変えています。前景色、背景色はまえとおなじです。行番号10のCOLORでは、周辺色の指定がカラーコード12ですから、周辺色は暗い緑にかわります。行番号50のCOLORでは、周辺色はカラーコード9ですから、明るい赤にかわります。

RUN

RUNは、パソコンにプログラムを実行させる

RUNは、パソコンに入力したプログラムを、実行させるコマンド、つまり命令です。

つぎのプログラムは、数あてゲームのプログラムです。

```
10 X=INT(RND(1)*10)+1
15 PRINT X
20 INPUT A
30 IF A>X THEN 60
40 IF A<X THEN 80
50 IF A=X THEN 100
60 PRINT "ﾓｯﾄﾁｲｻｲｶｽﾞﾀﾞﾖ"
70 GOTO 20
80 PRINT "ﾓｯﾄｵｵｷｲｶｽﾞｻ"
90 GOTO 20
100 PRINT "ｽｺﾞｲ!!ｱﾀﾘﾀﾞ":BEEP
110 END
```

このプログラムを入力したままでは、あなたとパソコンとで、数あてゲームをしようにもできません。なぜなら、プログラムを入力しただけでは、パソコンはプログラムを実行しないからです。

プログラムを実行させるには、RUNと入力します。RUNとは、「プログラムを実行しなさい」という意味の命令です。RUNを入力して、はじめて、パソコンはプログラムを実行するという、仕組みになっているのです。RUNを入力したらRETURNキーを押してください。するとパソコンは、プログラムを実行し始めます。

三角形を描くプログラム

```
10 S=1
20 FOR I=1 TO S
30 PRINT "O";
40 NEXT I
50 PRINT
60 S=S+1
70 IF S=13 THEN 90
80 GOTO 20
90  END

O
OO
OOO
OOOO
OOOOO
OOOOOO
OOOOOOO
OOOOOOOO
OOOOOOOOO
OOOOOOOOOO
OOOOOOOOOOO
OOOOOOOOOOOO
```

RNDは、不規則な数(乱数)を作りだす

パソコンには、不規則な数（乱数）を作りだす機能が、そなわっています。それがRND（X）です。RND（X）も、組み込み関数のひとつです。

```
10 X=RND(1)
20 PRINT X
30 GOTO 10
40 END
```

このプログラムを入力して、RUNを入力すると、行番号10のRND（1）が実行されて、乱数が作りだされ、それが左側のXに入れられて、PRINT　Xで表示されます。

```
run
.271279
.358892
.612562
.895349
.287716
.566827
```

もちろん、必ずこのような数が作りだされるとはかぎりません。そのつど、作りだされる乱数は異なります。これは1例です。

RND（X）のXに入れる数字は、1でも、2でも、3でもかまいません。ただ、RND（X）のXに0とか、－1、－2とかいった負の整数を与えると、たとえば、.306801なら.306801ばかりを作りだしてしまって、乱数を作りだしません。

RND（1）が作りだす乱数は、0から1の間の数です。

このRND（1）は、ゲームなどのプログラムによく使われますが、RND（1）で作られた乱数を、そのまま使うということは、まずありません。多くの場合、RND（X）で作りだされた乱数を加工して使っています。

① RND（1）が作りだす乱数

　　.000000から.999999

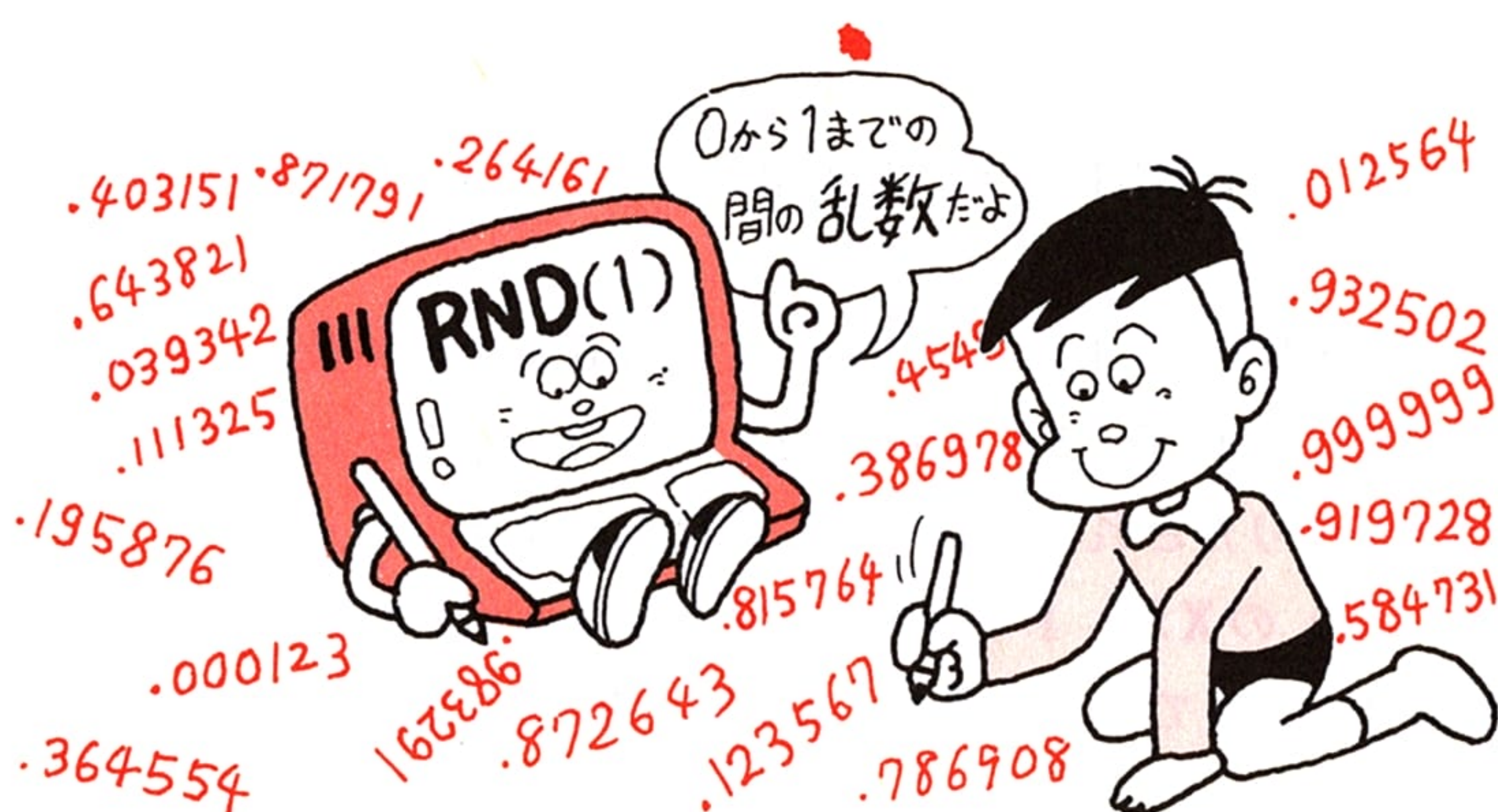
② 10＊RND（1）が作りだす乱数

　　0.00000 から9.99999

③ 10　RND（1）＋1が作りだす乱数

　　1.00000 から10.99999

④ INT（10＊RND（1））＋1が作りだす乱数

　　1から10

前頁の①、②、③、④についておはなしすると、

① RND（1）は、すでにおはなししたように、0から1までの間の乱数を作りだします。

② RND（1）で作り出された乱数を10倍すると、0から9.99999までの乱数になります。

③ RND（1）で作りだされた乱数を10倍し、それに＋1すると1.000000から10.99999 までの乱数になります。

④ RND（1）で作りだされた乱数を10倍し、それに＋1して、さらにINTで関数処理をすると、1から10までの乱数になります。INTは、INT(X)で表され、カッコのなかのXに小数点をもった数値を入れると、整数を算出します。

この①、②、③、④は、RND（1）で作りだされた乱数をどう加工すると、どういう具合になるか、その変化していく様子を示したものです。このうち乱数として、完成されたものは、①、②、③、④と加工を加えてきた最後の④です。

INT（10＊RND（1））＋1

この形で、ゲームなどのプログラムに使われています。もちろん

INT（6＊RND（1））＋1

INT（5＊RND（1））＋1

といったようにRND（1）を6倍したり、5倍したりする形もあります。

INT（6＊RND（1））＋1は、1から6までの整数で、乱数が作りだされます。INT（5＊RND（1））＋1は、1から5までの整数で、乱数が作りだされるといったことになります。

RND（X）のXが、1ではなく、2とか3が使われていることもままありますが、1でも、2でも、3でも、RND（X）の働きには変りありません。

READ〜DATA

READ〜DATAは、データの量が多いとき使うと便利

パソコンにデータを入力することばとして、36頁で、INPUTについておはなししました。

INPUTは、パソコンがプログラムの実行に移ってから、キーボードのキーを叩いて、INPUTのあとにつづく変数のデータを入力してやるといったものでした。

READ〜DATAも、パソコンにデータを入力することばですが、READ〜DATAはあらかじめプログラムのなかに、データを書き込んでおくといった方法をとります。

```
10 READ A,B
20 C=A*B
30 PRINT C
40 GOTO 10
50 DATA 5,7,9,3,2,6,8,1,4,10
60 END
```

うえのプログラムは、READ〜DATAを使った掛け算のプログラムです。

行番号10のREADということばのあとに、A、Bといった変数があるように、READのあとには、必ず変数を書きます。これで、READの意味は、READのあとにつづく変数A、Bのデータを読みとりなさいということになります。

READが、そのあとにつづく変数に読みとるデータは、DATAということばのあとに書かれます。

行番号50のDATAということばのあとに書かれた数値が、READ A、BのA、Bに読みとられるデータです。このDATAに示された数値は、一番最初から、順番に読みとられていきます。読みとられ方は、あとでおはなしします。

READとDATAは、このようにいつもいっしょに使われます。

READ～DATAは、変数の値をいろいろに変えて、いろいろな計算結果を求めるときや、データの量が多い、といったときに使うと、便利です。

では、このプログラムをパソコンに入力して、実行させて、READ～DATAの働きをみてみましょう。

RUNを入力すると、パソコンは、行番号10のREAD　A、Bを実行します。READ　A、Bは、AとBのデータを、行番号50のDATAに示されている値から読みとりなさいということです。

したがって、まず一番最初の5とつぎの7が、AとBにそれぞれ読みとられます。Aに読みとられた5と、Bに読みとられた7は、行番号20のAとBに送られ、A×Bの計算式で計算されます。

A×Bの計算結果35は、＝の左側のCに入れられ、行番号30のPRINT Cに送られます。

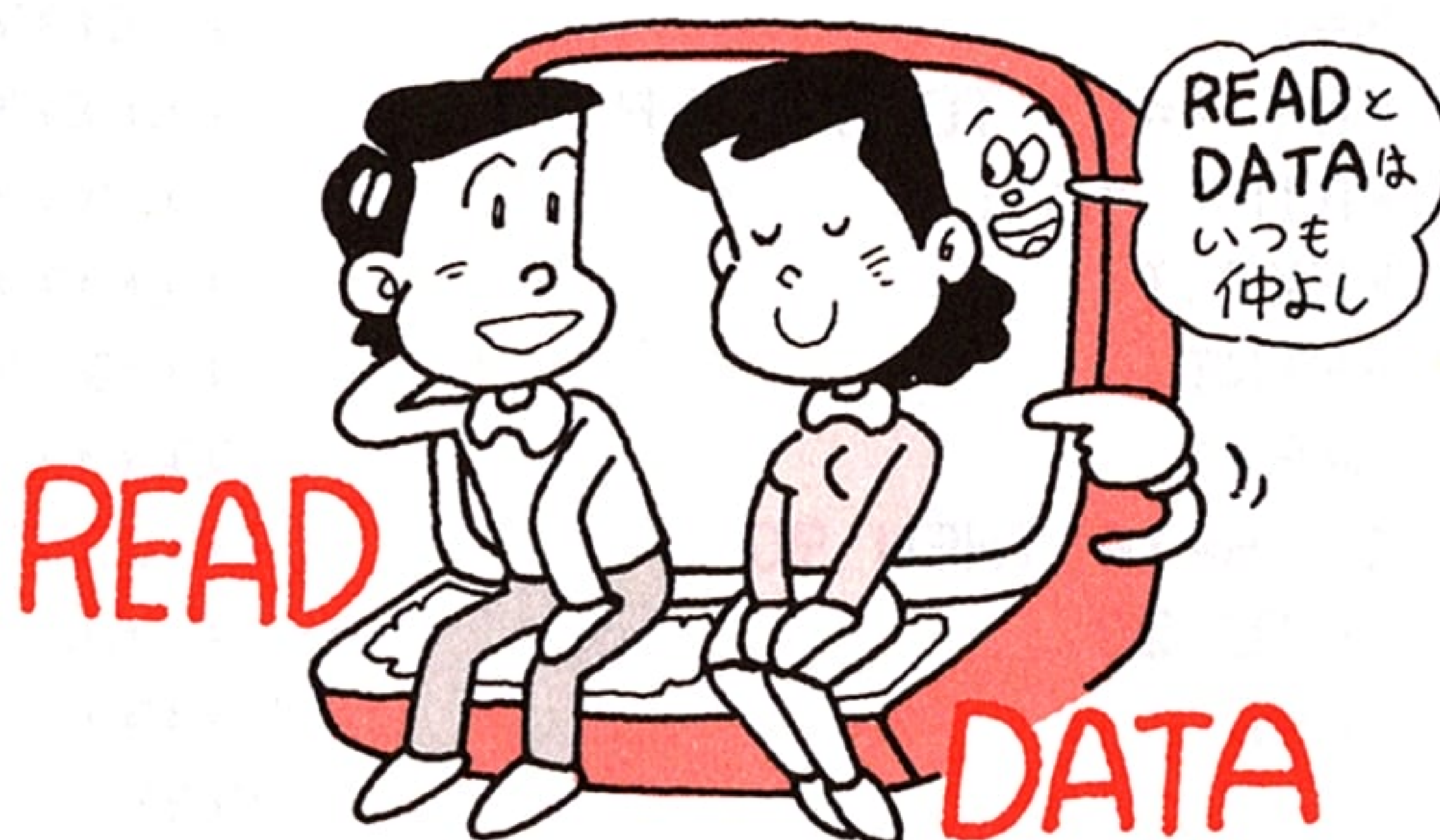
行番号30のPRINT　Cは、Cの中身を表示しなさいということですから、Cの中身35が、ディスプレイに表示されます。

Cの中身35をディスプレイに表示すると、つぎに行番号40の GOTO　10を実行します。GOTO　10は、行番号10に飛びなさいということです。ここにGOTOを使わないと、35をディスプレイに表示しただけで、そのあとにつづいているデータの処理が行われません。GOTO 10として、パソコンの実行を行番号10にもどすことによって、ふたたびREAD　A、Bが実行されて、つぎのデータが読みとられるのです。

```
10 READ A,B

40 GOTO 10
```

GOTO　10で、パソコンの実行は行番号10にもどります

GOTO　10で、行番号10に戻り、ふたたびREAD　A、Bが実行されます。2回目のデータの読みとり方は、1回目で読みとられたデータのつぎのデータです。

READ～DATA

```
10 READ A,B
```

2回目のデータの読みとり方は、1回目で読みとられたデータのつぎのデータです

```
50 DATA 5,7,9,3,2,6,8,1,4,10
```

パソコンでは、DATAのあとに書かれたデータは、1度使われると、つぎには利用できない仕組みになっているのです。

READ A、Bに読みとられた9と3のデータは、行番号20のA×Bで計算されて、その計算結果27は、＝の左側のCに入れられます。そして、行番号30のCに送られて、PRINT　Cで27が表示されます。

このことがくり返されて、2×6、8×1、4×10の計算が行われます。

最後のデータ4と10を読みとって、4×10の計算を終了しても、パソコンは、行番号60のENDを実行しません。なぜなら、このプログラムでは、GOTOを使って、くり返し処理を行っているからです。4×10を実行すると、GOTO　10によって、行番号10に戻ります。そしてREAD　A、Bを実行します。ところが読みとるデータが、もう行番行50のDATAにはありません。そこで

```
Out of DATA in10
```

三角形を書くプログラム

```
10 S=1
20 FOR I=13 TO S STEP -1
30 PRINT "○";
40 NEXT I
50 PRINT
60 S=S+1
70 IF S=14 THEN 90
80 GOTO 20
90  END
```

```
○○○○○○○○○○○○○
○○○○○○○○○○○○
○○○○○○○○○○○
○○○○○○○○○○
○○○○○○○○○
○○○○○○○○
○○○○○○○
○○○○○○
○○○○○
○○○○
○○○
○○
○
```

行番号10のREADで読みとるデータが、DATAのなかにないというエラーメッセージを表示して、実行を終了します。つぎが実行結果です。

```
run
 35
 27
 12
 8
 40
Out of DATA in 10
```

ところで、このプログラムでは、READ～DATAをひとつ使いましたが、READ～DATAは、おなじプログラムのなかに何個使ってもかまいません。

```
10 READ A,B
20 READ C,D
30 S=A*B
40 T=C/D
50 PRINT S,T
60 DATA 5,10
70 DATA 5,10
80 END
run
 50            .5
```


ただ、このような場合は、必ずREAD～DATAをひと組づつ使うようにしなければなりません。

また、DATAに示すデータの数値がおなじである場合、データの数値がおなじだからといって、DATAをひとつにまとめてしまってはいけません。DATAのあとにつづく、データは、１回しか使えないので、

READ～DATA

たとえデータの数値がおなじであっても、うえのプログラムのように、READのあとにある変数の数だけ書かなければなりません。そうしないと、Ouf　of　DATA　in20というエラーメッセージが表示されます。

READ～DATAのかたち

READ　変数、変数、………、変数
　　∫
DATA　定数、定数、………、定数

TABを使ったプログラム

```
10 FOR X=1 TO 16
20 PRINT TAB(X*X/10);X
30 NEXT X
40 END

 1
 2
 3
  4
   5
    6
     7
        8
          9
            10
              11
                12
                  13
                     14
                        15
                           16
```

RETURNキー

RETURNキーは、プログラムをメモリに記憶させる

パソコンに仕事をさせるためには、プログラムを入力しなければなりません。

プログラムは、パソコンにやらせる仕事の手順を示したもので、機械語、アセンブリ言語、フォートラン、BASICといったプログラミング言語（プログラムを作ることば）で書かれます。

パソコンのプログラムは、ほとんどの場合、BASICで書かれます。つぎのプログラムが、BASICで書かれたプログラムです。

```
10 READ A,B,C,D,E
20 F=A+B+C+D+E
30 F=F/5
40 PRINT "ﾍｲｷﾝﾁ";F
50 DATA 52,64,93,27,15
60 END
run
ﾍｲｷﾝﾁ 50.2
```

このプログラムは、パソコンに入力されて、パソコンのなかのメモリに記憶されてから、処理されます。いいかえれば、メモリに記憶されないものは、処理されないということです。

ふつう、プログラムの入力は、キーボードのキーをたたいて、1行分づつ入力していきます。

10 READ A, B, C, D, E

この場合、キーボードのキーをたたいただけでは、ディスプレイの画面のうえに、これらの文字が表示されていたとしても、それはただ表示されているだけで、メモリのなかに記憶されているわけではありません。10 READ A、B、C、D、Eと、ひとつの文を入力したら、必ず、RETURNキーを押します。これではじめてメモリのなかにひとつの文が記憶されたことになります。

ディスプレイの画面のうえでみると、それまでEのうしろで点滅していたカーソルは、行を変えて、次の行のはじめのところに移動します。このため、RETURNキーの働きは、改行のキーととらえられがちですが、RETURNキーのもっとも大切な働きは、入力した文を、パソコンのメモリのなかに、記憶させるということにあるのです。

RETURNキーの押し忘れといったことは、プログラムを入力するときには、ほとんど生じませんが、プログラムを修正したりしたときに、よくRETURNキーを押し忘れることがあります。プログラムを修正したときも、RETURNキーを押さないと、修正したことがメモリの中に記憶されず、もとのままですから、注意してください。

RESTORE

RESTOREを使うと、データを繰り返し使うことができる

181頁でおはなししたREAD～DATAの項で、DATAのあとに示されたデータは、1度使ってしまうと2度と使うことはできないとおはなししました。

```
10 READ A,B
20 READ C,D
30 S=A*B
40 T=C/D
50 PRINT "ｶｹｻﾞﾝ=";S,"ﾜﾘｻﾞﾝ=";T
60 DATA 5,10
70 DATA 5,10
80 END

run
ｶｹｻﾞﾝ= 50        ﾜﾘｻﾞﾝ= .5
```

そして、たとえデータの数値がおなじであっても、うえのプログラムのように、READのあとにある変数の数だけ書かなければならないとも、おはなししました。

ところが、RESTOREを使うと、ひとつのデータを2度でも、3度でも使うことができるようになります。

つまりパソコンは、RESTOREということばに出会うと、RESTOREのつぎの行番号のREADのあとの変数に、1度使ってしまった

データを、ふたたび順番に読みとらせるのです。

```
10 READ A,B
20 RESTORE
30 READ C,D
40 S=A*B
50 T=C/D
60 PRINT "ｶｹｻﾞﾝ=";S,"ﾜﾘｻﾞﾝ=";T
70 DATA 5,10
80 END
run
ｶｹｻﾞﾝ= 50       ﾜﾘｻﾞﾝ= .5
```


したがって、まえのプログラムにRESTOREを使うと、うえのプログラムのようにDATAは、ひとつですむようになります。

これは、行番号10のREAD　A、Bで1度読みとられたデータが、RESTOREの働きによって、RESTOREのつぎの行番号30のREAD　C、Dにも読みとられるようになるからです。

また RESTORE のあとに行番号を書いて、DATAを指定すると、その指定されたDATAに示されているデータが、ふたたび読みとられます。

```
10 READ A,B,C,D,E,F,G
20 RESTORE
30 READ H,I
40 RESTORE 100
50 READ J,K,L
60 PRINT A;B;C;D;E;F;G
70 PRINT H;I
80 PRINT J;K;L
90 DATA 1,2,3,4
100 DATA 5,6,7
110 END
run
 1  2  3  4  5  6  7
 1  2
 5  6  7
```

行番号90、100のDATAから1、2、3、4、5、6、7のデータを読みとる

RESTOREの働きでデータの一番最初にもどって、1、2を読みとる

RESTORE 100の働きで行番号 100のDATAから5、6、7のデータを読みとる

このプログラムでは、行番号40のRESTOREのあとに、読みとるべきDATAの行番号が100として、指定されています。

この読みとるべきDATAの行番号100が指定されていないと、行番号20のRESTOREの働きで行番号30のREAD　H、Iが、データの一番最初にもどって、1　2とデータを読みとったのとおなじように、行番号50のREAD　J、K、Lも、1　2　3とデータの一番最初にもどって、読み込んでいってしまいます。

しかし、行番号40のRESTOREは、RESTORE 100となっていることから、行番号50のREAD　J、K、Lは一番最初のデータにもどって、データを読みとるということはしないで、指定された行番号100のDATA　5、6、7を読みとっていくということになります。

LET

LETは、変数の値を決める

いま私たちがBASICでプログラムを書くとき、A＝3、B＝2、C＝A＋Bなどといったステートメントには、次のプログラムのように、LETをつけて書かなくなりました。

```
10 LET A=3
20 LET B=2
30 LET C=A+B
40 PRINT C
50 END
```


このように、LETをひとつひとつのステートメントにつけて書くと、プログラムがLETだらけになってしまって、見た目にうるさいプログラムになってしまうからかもしれません。

理由はともかく、いまのパソコンのほとんどは、LETをつけなくても、通用するようになっています。

```
10 A=3
20 B=2
30 C=A+B
40 PRINT C
50 END
```


ただこれは、LETを省略してもよいということだけであって、LET

の働きそのものが、なくなってしまったわけではありません。

私たちが、A＝2とパソコンに入力すると、パソコンはA＝2にLETを補って、LET　. A＝2にして受け入れているととらえていたほうが、たとえLETを書かなくても、間違いのないプログラムを作ることができるでしょう。

では、LETとはどういうものなのでしょうか。

```
10 LET A=3
20 LET B=2
30 LET C=A+B
```

これは、まえのプログラムから、LETを必要とするステートメントだけを、抜きだしたものです。

この場合のLETは、〝＝〞の左側にある変数AやBに、〝＝〞の右側にある2や3といった値を入れなさいということです。変数は、2とか3のように、決ったものではありませんから、LETによってAに2を入れ、Bに3を入れて、Aは2である、Bは3であるといったように、決めてしまうわけです。

もちろん、C＝A＋Bも同じです。AとBをたして、その和をBに入れなさい。そして、Cをその値にしなさいということです。

このように考えると、A＝2、B＝3、C＝A＋Bの〝＝〞という記号は不思議なものに映りますが、この場合の〝＝〞は、等しいという意味ではありません。〝＝〞記号は、左向きの矢印〝←〞を意味するものと考えてよいでしょう。次のようにです。

```
10 LET A←3
20 LET B←2
30 LET C←A+B
```

この矢印は、必ず左を向いていて、〝→〞といったように右向きにはなりません。常に〝←〞このままです。

そのため、LETの場合の変数は、必ず〝＝〞記号の左側になければなりません。

これを、20　3＝B、40　A／B＝Cといったように、〝＝〞の左右を入れ違えて入力すると、パソコンは、Syntax　errorを表示します。

さて、LETの場合、もうひとつ注意しなければならないことがあります。それは、30　LET　C＝A＋Bといったように、〝＝〞記号の右側に、変数による計算式がきたときです。

```
10 LET A=2
20 LET B=3
30 LET C=A+B
40 PRINT C
50 END
run
 5
```

AとBの値が、そのまえの行番号で決められています

この場合は、C＝A＋BのAとBの値は、行番号10と行番号20で、A＝2、B＝3と決められていますから、AとBに2と3が送られてきて、パソコンは、A＋Bの結果の5を表示します。

では次の場合は、どうでしょうか。

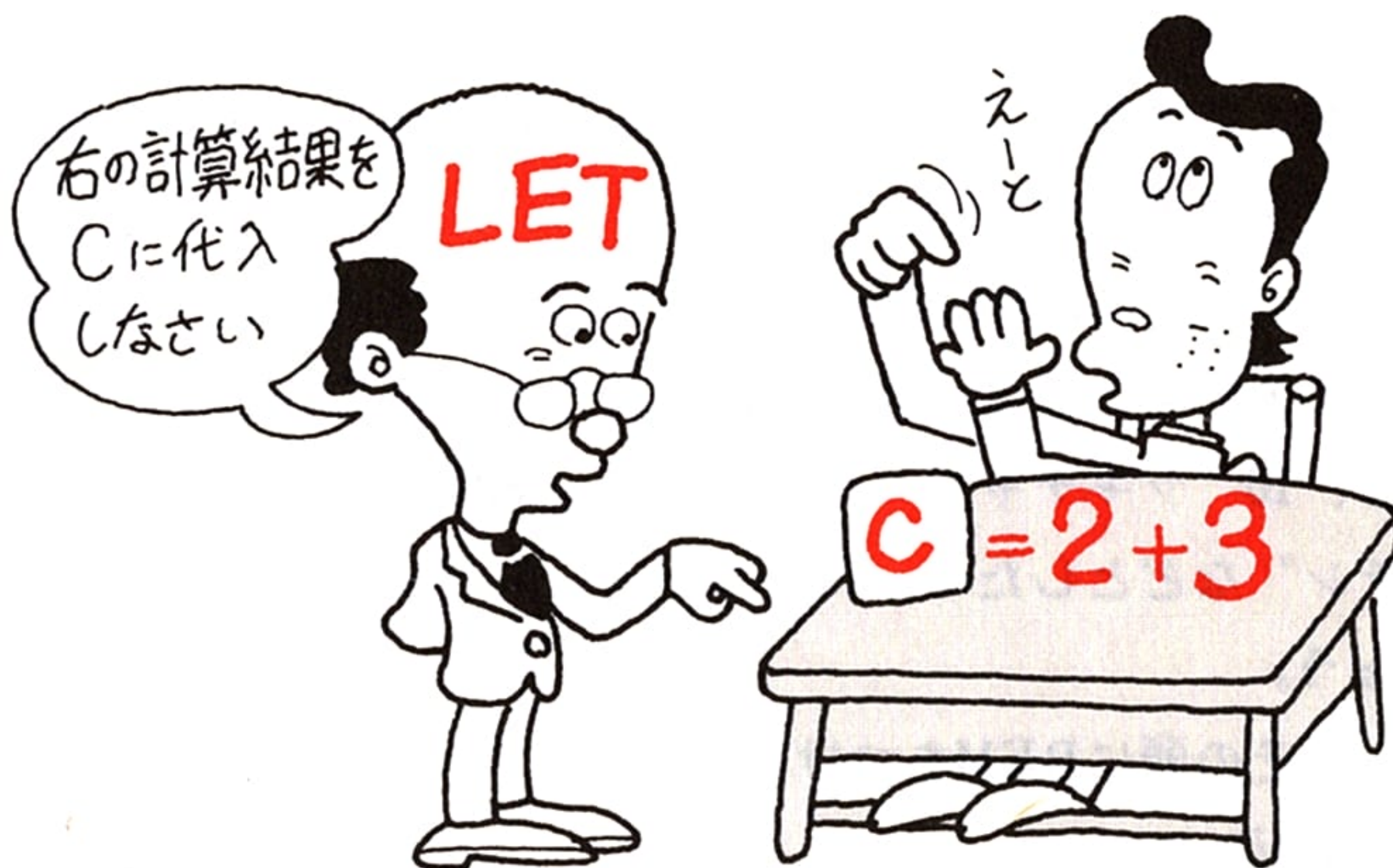
```
10 LET A=2
20 LET D=3
30 LET C=A+B
40 PRINT C
50 END

run
 2
```

変数Bの値が、そのまえの行番号で決められていないので、0とみなされます

Cの中身は2なので、2が表示されます

この場合は、C＝A＋Bのうち、Aの値だけが行番号10でA＝2と決められていて、Bの値はその前の行番号のどこにも決められていません。このようなとき、Bは0とみなされて、パソコンは、2を表示します。また、行番号20のD＝3は、メモリに記憶されたままで終ります。

このように、〝＝〟の右側に変数による計算式がきた場合、計算式のなかの変数の値が、それ以前の行番号のどこかで決められていないと0とみなされますから、注意しなければなりません。

REM

REMは、タイトルや説明の頭につける

REMは、REMARKといったことばの略で、注釈といった意味です。BASICで書くプログラムのなかに使うREMも、この意味で使います。

```
10 REM "ﾂｷﾁｬｸﾘｸｹﾞｰﾑ"
800 REM "ﾘｭｳｾｲｸﾞﾝﾉｻﾌﾞﾙｰﾁﾝ"
```

このようにREMは、プログラムのタイトルや、プログラムのなかで説明しておいたほうがよいところなどの説明文の頭につきます。

ただ、パソコンのメモリのなかに記憶できる記憶容量を考えたとき、むやみにREMを使うことは、感心したことではありませんが、REMを使ってタイトルをつけたり、プログラムのなかに説明をつけ加えておくと、あとになって、プログラムをみたとき、なんのプログラムであるかすぐわかりますし、忘れてしまったプログラムの働きをも思い出すことができて便利です。

したがって、適当にREMを使って、タイトルや説明をつけ加えておくことも必要です。要は、ほどほどに使うということです。

これを、10 〝ツキチャクリクゲーム〞、800 〝リュウセイグンノサブルーチン〞などとしたら、パソコンは、エラーメッセージを表示するでしょう。

しかし、その頭にREMをつけると、パソコンは、これは注釈だなと判断して実行しません。

LOCATE

LOCATEは、文字などを表示する位置を指定する

文字などを、ディスプレイの画面のある位置に、表示させるときに使うことばとしては、128頁のTAB（X）や、113頁のSPC（X）などといったことばがあります。

これらのことばとおなじように、ディスプレイの画面のある位置に文字を表示させることばとして、LOCATEということばがあります。このLOCATEは、非常に便利なことばで、よくプログラムのなかに使われています。

```
10 LOCATE 10,7
20 PRINT "モグ゛ラタタキゲ゛ーム"
30 END
```


行番号10のLOCATEのあとにつづいている10と7といった数字は、10が列番号、7が行番号です。このようにLOCATEのあとには、列番号と、行番号を書きます。

こうすると、カーソルが列番号、行番号で指定された位置まで移動して、移動したカーソルの位置から、つぎの行番号のPRINTで示された文字が表示されることになります。

このLOCATEを使ったときは、カーソルは文字を表示する位置には現われません。

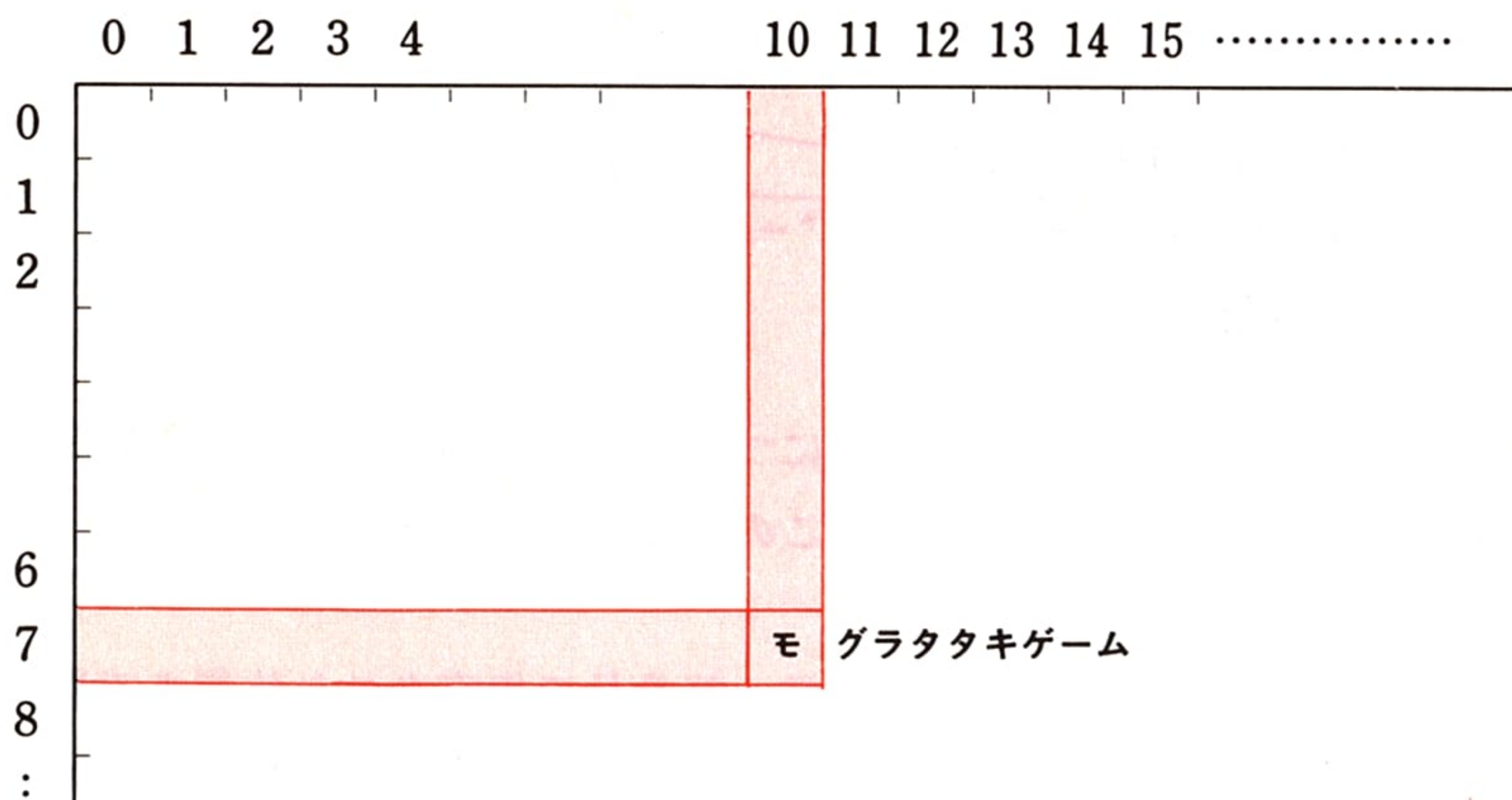
うえのプログラムの場合は、パソコンをスイッチオンしたとき、自動的にセットされる画面表示にもとづいて、列番号、行番号を指定し

ています。

パソコンをスイッチオンしたときの画面表示は、列番号が0から28、行番号が0から23です(46頁参照)。

WIDTH指定をして、WIDTH 32とすると、列番号は0から31となりますから、それにもとづいてLOCATEのあとの列番号を指定しなければなりません。

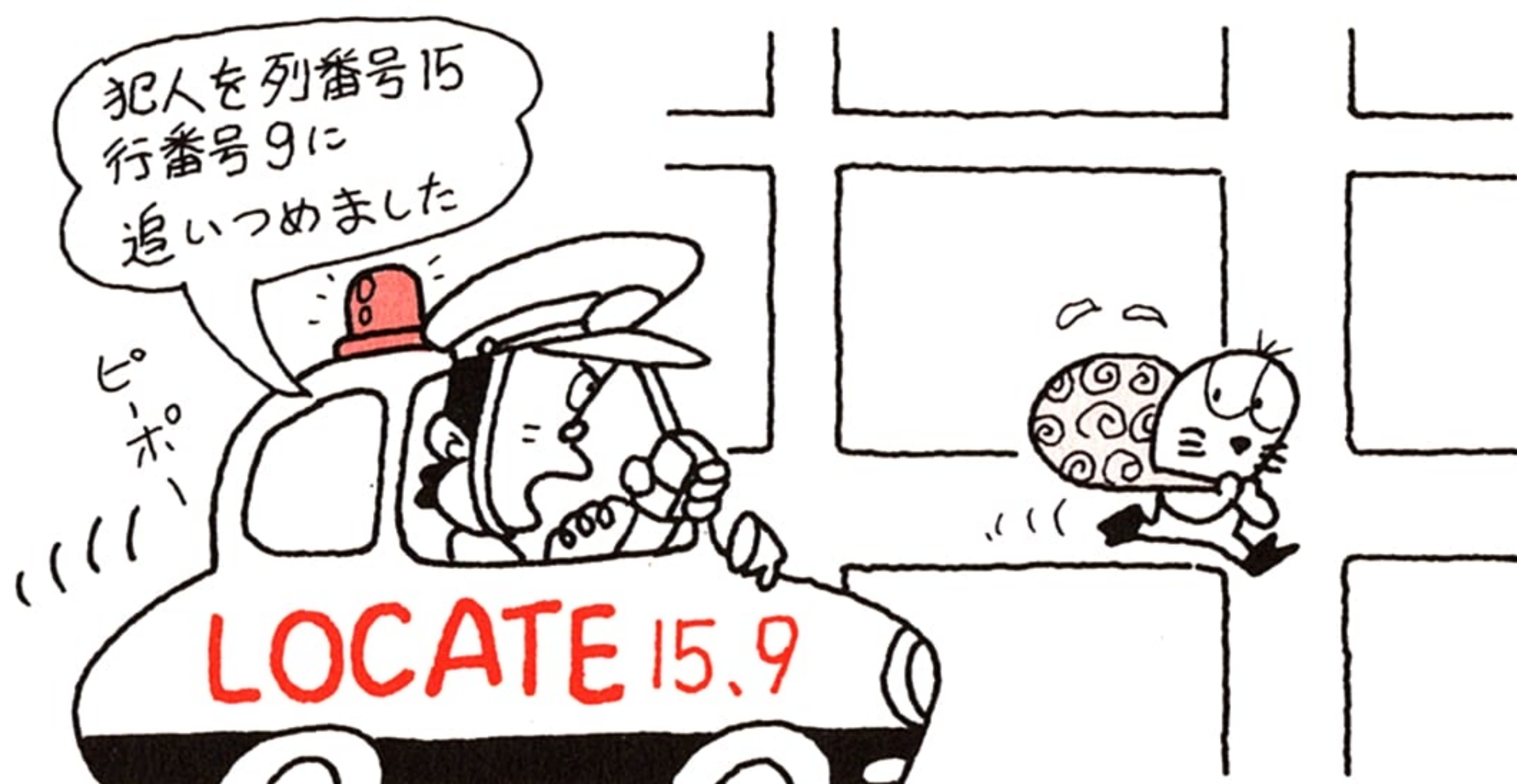
さて、このプログラムを入力して、実行させると、行番号10のLOCATE 10, 7が実行されて、カーソルはつぎの図に示すように、列番号10と行番号7の交わる位置にまで移動します。

そして行番号20のPRINT 〝モグラタタキゲーム〟が実行されると、モグラタタキゲームの「モ」が、LOCATE 10, 7の指定で移動したカーソルの位置に表示され、つづいて、そのあとの文字が表示されることになります。

このプログラムの場合、モグラタタキゲームという文字は、画面のほぼまんなかあたりに表示されます。

さて、LOCATEを使って、文字などを、ある位置に表示させるだけなら、これでよいのですが、文字などを表示させたままでは、その文字がじゃまになってしまうことがあります。

そのときに、LOCATEで表示した文字を消す方法について、おはなししておきましょう。

```
10 LOCATE 10,4
20 PRINT "キョウノテンキ"
30 FOR I=1 TO 1500
40 NEXT I
50 CLS
60 LOCATE 10,4
70 PRINT "アメ ノチ クモリ"
80 FOR I=1 TO 1500
90 NEXT I
100 CLS
110 END
```

行番号10のLOCATE　10，4が実行されると、カーソルが列番号10、行番号4のところに移動します。そして行番号20のPRINT〝キョウノテンキ〟が実行されると、キョウノテンキがカーソルのある位置から表示されます。10と4の位置は、画面のまんなか、うえのほうです。行番号30と40にFOR～NEXTを用いたのは、画面表示を消すCLSを行番号20につづけて使うと、表示された文字が一瞬のうちに消されてし

まって、なにを表示したのかわからないからです。そのため、CLSのまえにFOR～NEXTを入れてウエイト・ループとして、Iが初期値の1から最終値の1500になるまで繰り返えさせて、CLSの働きを遅らせているわけです。したがって、その間は、キョウノテンキが表示されています。

FOR～NEXTのIが最終値の1500になると、ウエイト・ループから抜け出して、行番号50のCLSが実行されて、キョウノテンキという文字が消されます。

つぎに行番号60のLOCATE 10, 4が実行され、つづいて、行番号60のPRINT "アメノチクモリ"が実行されると、こんどは、その位置に、アメノチクモリが表示されます。

行番号80、90、100は、行番号30、40、50とおなじように働きます。

KEY OFFとKEY ON

KEY OFFは、ファンクションキーの表示を消す
KEY ONは、ファンクションキーの内容を表示する

パソコンに電源を入れると、画面の一番下にcolor、auto、goto、list、runといったファンクションキーの内容が表示されます。ゲームプログラムなどを実行させるとき、このファンクションキーの内容の表示がじゃまになります。これを消すときに使うのが、KEY OFFです。プログラムの最初のほうで、

```
10 KEY OFF
```
←ファンクションキーの表示を消す

とします。このようにすると、ファンクションキーの表示が消されます。この場合は、ただファンクションキーの内容の表示が消されるだけです。ふつう、ゲームプログラムなどを実行させるときは、ファンクションキーの内容の表示はもちろん、画面に表示されているものすべてを消してから実行させますから、そのときは、

```
10 CLS:KEY OFF
```
←CLSで画面を消去し、ファンクションキーの表示を消す

というように、CLSのあとをコロン「：」で区切って、そのあとにKEY OFFを置きます。

一度、KEY OFFを実行すると消えたままになりますから、ファンクションキーの内容を表示させるときは、ダイレクトモードで、

```
KEY ON
```
←RETURNキーを叩く

と入力して、RETURNキーを叩きます。

SCREEN

SCREENは、画面モードの設定に使う

SCREEN(スクリーン)は、画面モードを設定するときに使います。画面モードとは、テキストモードとかグラフィックモードのことです。テキストモードは、パソコンに電源を入れたときの状態で、プログラムやデータなどを入力したり、文字や計算結果などを表示する画面ですから、CIRCLEやLINEなどのグラフィック命令を使って線を引いたり、図形を描いたりすることができません。これに対して、グラフィックモードは図形などを描く画面ですから、LINE、PSET、CIRCLE などのグラフィック命令を使って、線や図形を描くことができますが、プログラムやデータを入力したり、文字や計算結果を表示させたりすることができません。

▶テキストモードの指定

まず、テキストモードの指定です。パソコンに電源を入れたときはテキストモードになっています。したがってSCREENを使って、ふたたびテキストモードにする必要などないと思うかもしれませんが、SCREEN でテキストモードを指定することによって、パソコンの持っている能力を十二分に使うことができるようになります。

SCREENで、テキストモードを指定するには、つぎのように、SCREENのあとに0か1を指定します。

```
SCREEN  0
SCREEN  1
```

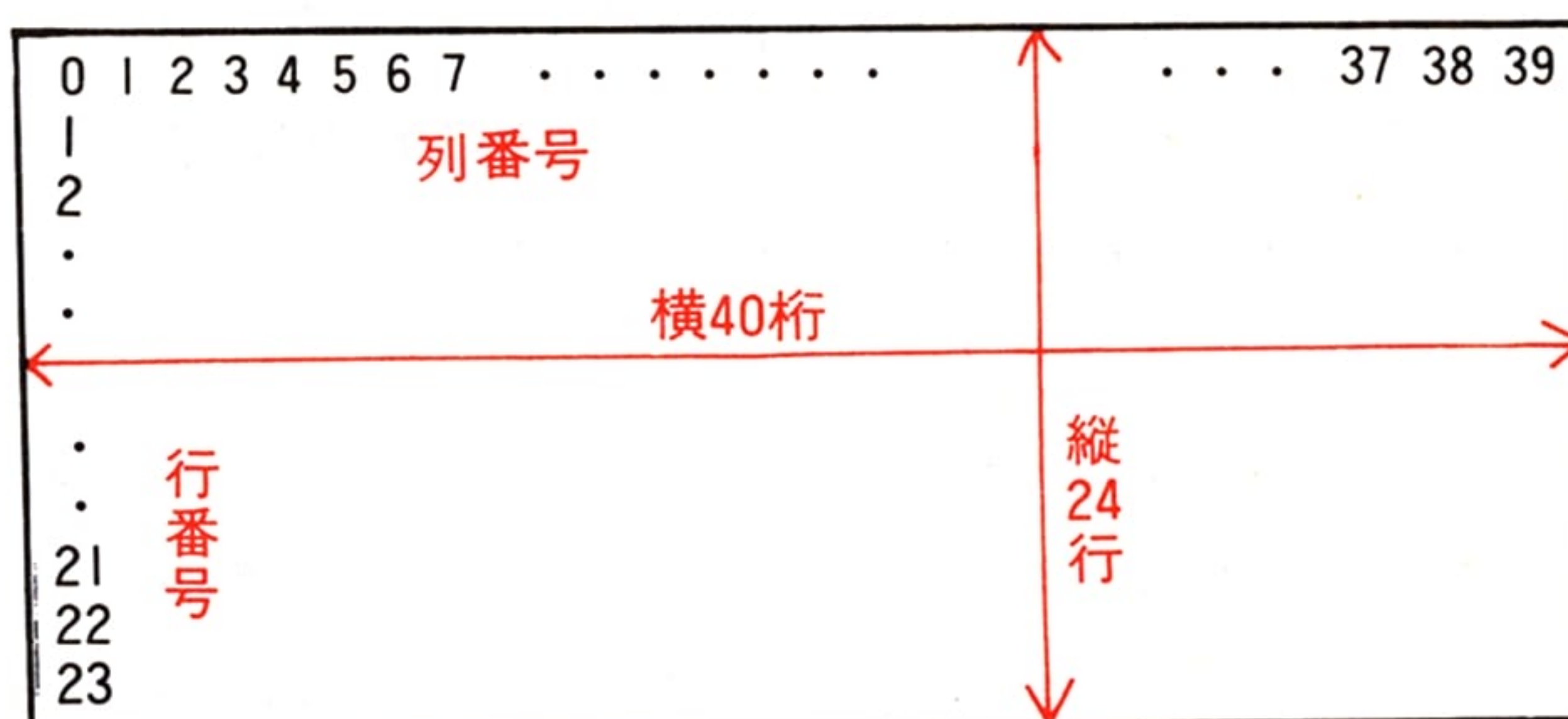
テキストモードの指定

SCREEN 0： SCREEN 0と指定すると、横1行に40桁まで文字

を表示することができるようになります。もし、SCREEN　0と指定して、横1行に40桁の文字を表示できないときは、

WIDTH　40

と指定すると、横1行に40桁の文字を表示することができるようになります。

SCREEN　0で横1行40桁表示にしたときは、列番号は0から39になります。縦の行数は変更することができませんから、行番号はパソコンに電源を入れたときとおなじ、0から23までです。

SCREEN　1：SCREEN　1と指定すると、横1行に表示することができる文字の桁数は32桁までとなります。つまり、SCREEN　0と比べると、表示することができる文字の桁数が減ることになります。縦の行数は変わりませんから、24行です。横1行32桁表示のときの列番号は、0から31となります。

SCREEN　1もテキストモードなので、グラフィック命令を使って線を引いたり、図形を描いたりすることはできませんが、スプライト機能（229頁参照）を使うことができます。

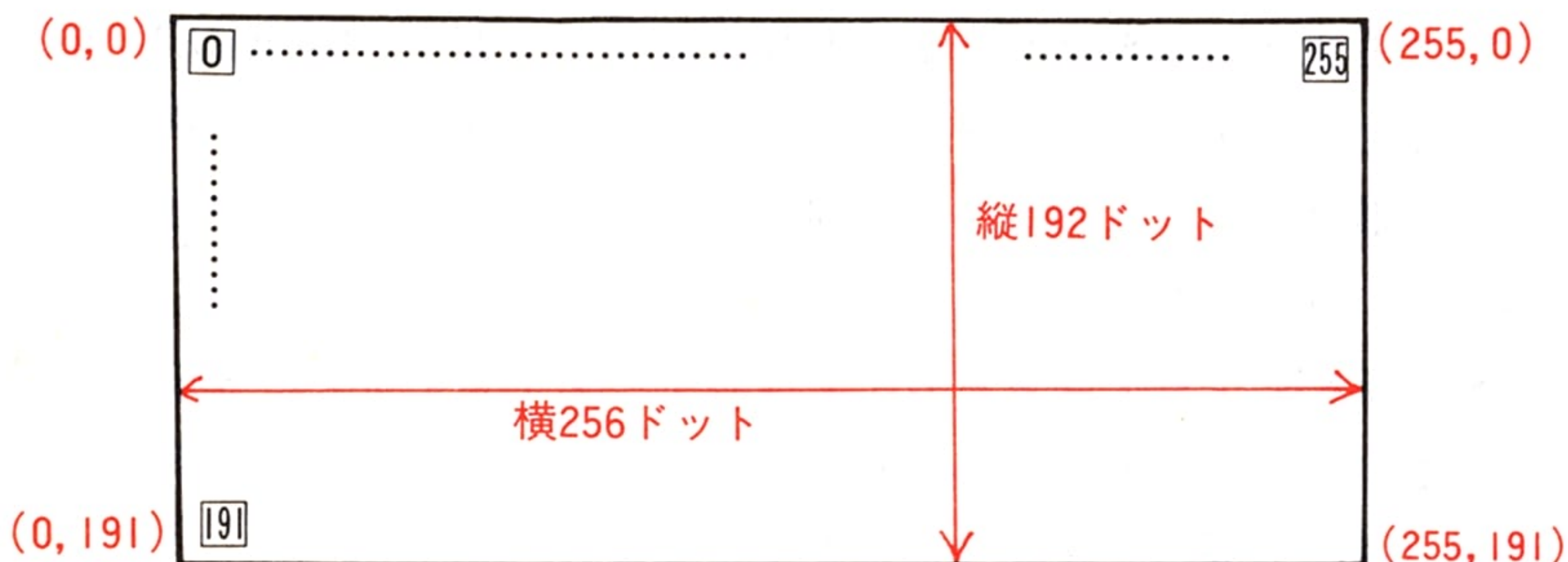
テキストモード	表示桁数	スプライト機能	グラフィック命令
SCREEN　0	最大40桁	使えない	使えない
SCREEN　1	最大32桁	使える	使えない

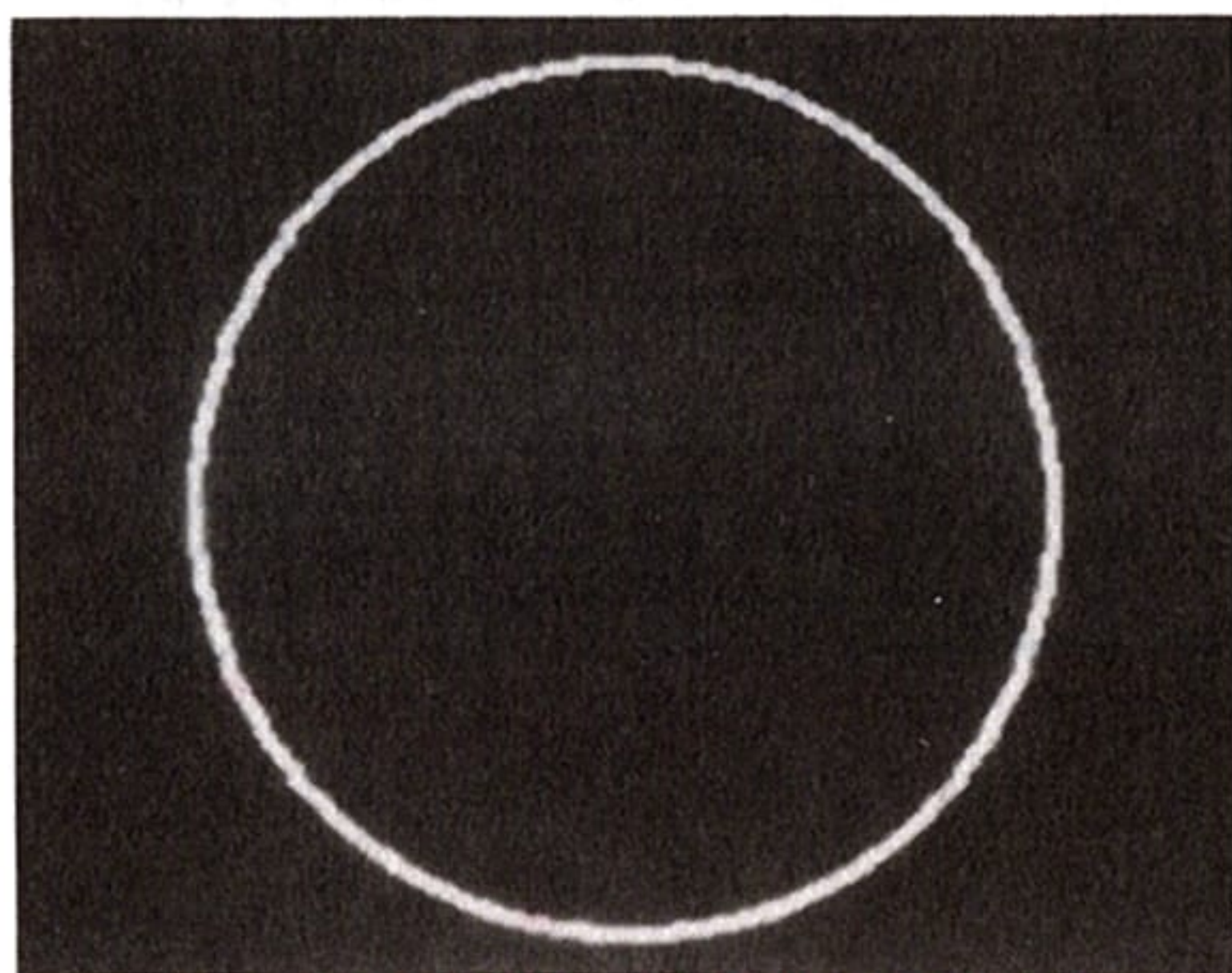
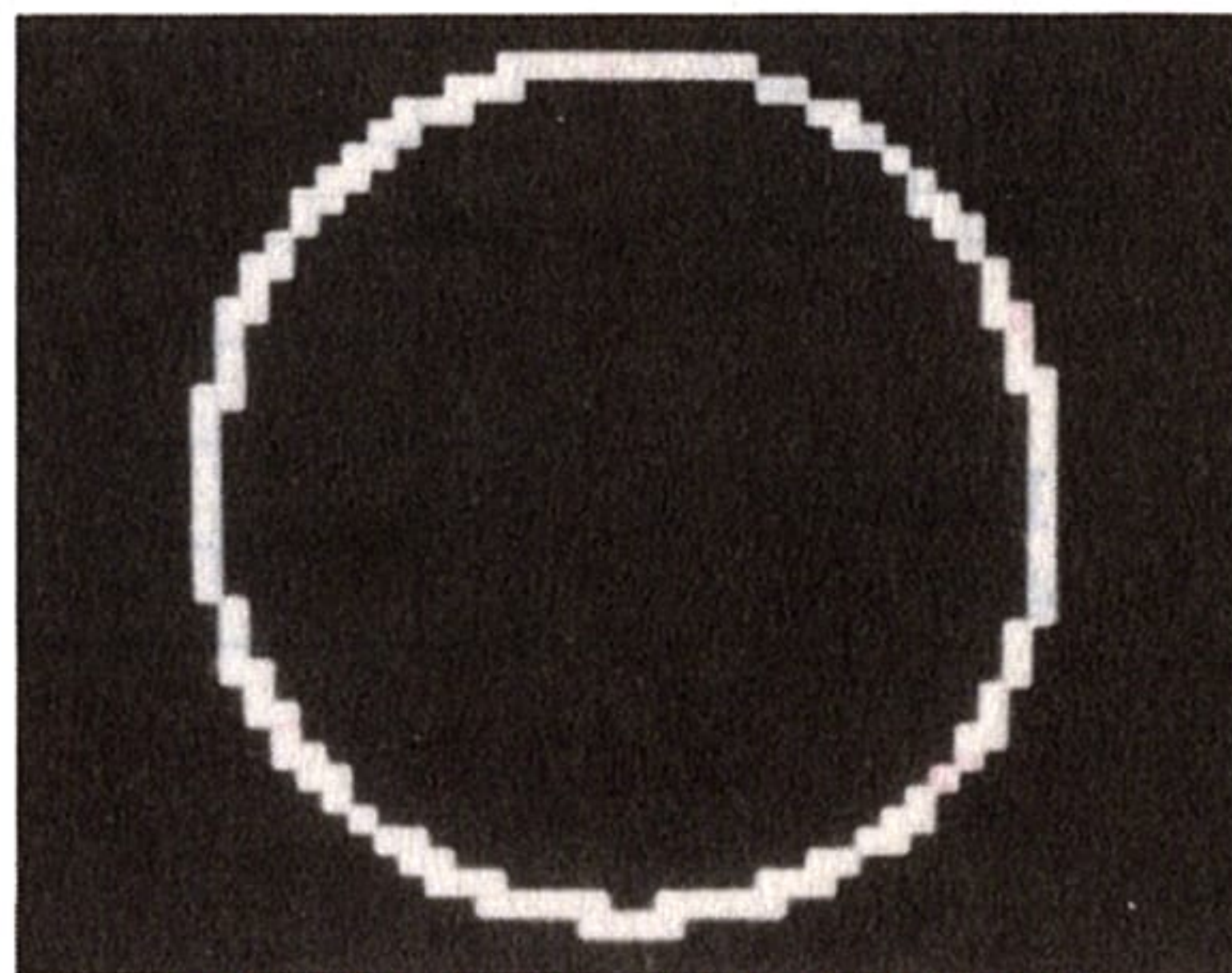
▶グラフィックモードの指定

図形などを描くグラフィックモードを指定するには、SCREEN のあとに、2か3を指定します。

```
SCREEN  2
SCREEN  3
```
←グラフィックモードの指定

SCREEN 2：SCREEN 2の指定は、**高解像度グラフィックモード**といいます。高解像度グラフィックモードは、ドット（点）で円や図形などを描くので、SCREEN 3の指定に比べると、細かい線で円や図形を描くことができます。SCREEN 2の指定でどのように細かく円や図形を描くことができるかは、あとでSCREEN 2の指定で描いた円の写真と、SCREEN 3の指定で描いた円の写真とを一緒に示しますから、比較してみてください。

ところでグラフィックモードの画面は、テキストモードの画面と違って、グラフィック座標となります。グラフィック座標は、つぎに示すように、列番号は、0から255となります。行番号は、0から191となります。

グラフィックモードでは、円を描く位置や図形などを描く位置の指定を、このグラフィック座標で行います。グラフィック座標を使った位置の指定の仕方は、LINEやCIRCLEなどのグラフィック命令の項を参照してください。

SCREEN　3：SCREEN　3の指定は、**マルチカラーモード**といいます。マルチカラーモードは、SCREEN　2の指定による高解像度グラフィックモードと違って、ドット単位で円や図形などを描くのではなく、16ドットを1ブロックとしたブロック単位で描きます。したがってつぎの写真に示すように、SCREEN　3の指定で描いた図形は、非常に荒く描かれることになります。

SCREEN　2で描いた円

SCREEN　3で描いた円

▶スプライトサイズの指定

MSX・パソコンには、0から31番までのスプライト面というものがあります。スプライト面には、自分で作ったキャラクタなどを表示させることができます。もちろん、表示するばかりでなく、キャラクタを移動させることもできます。

SCREEN　1、SCREEN　2、SCREEN　3を指定したときは、このスプライト機能を使うことができます。

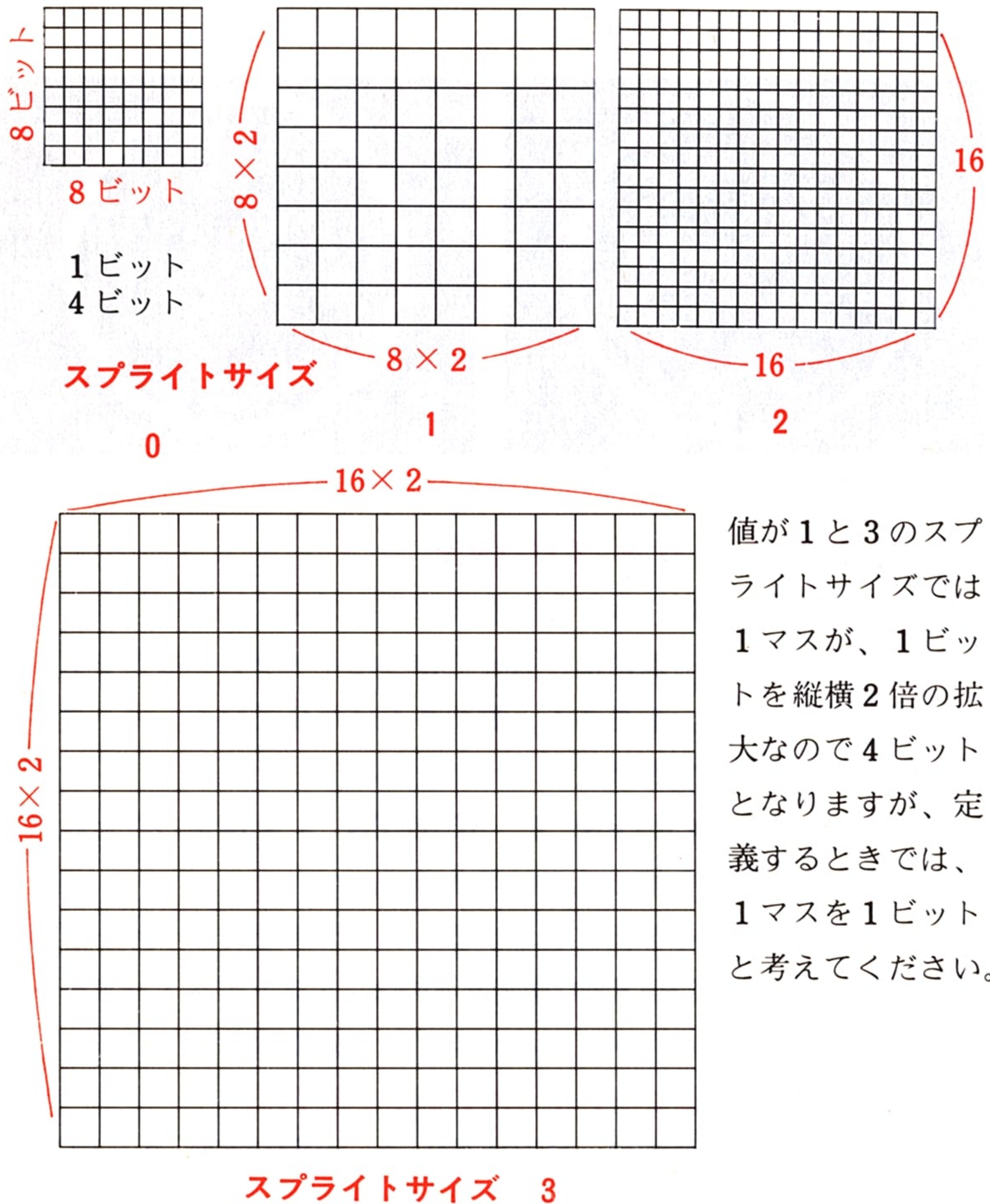
ところで、スプライト面に表示するキャラクタをスプライト・パターンといいます。SCREENで指定するのは、スプライト・パターンがどのくらいの大きさなのか、その大きさの指定です。

SCREEN 〈画面モード〉,〈スプライトサイズ〉

〈画面モード〉の指定は、SCREEN　1、SCREEN　2、SCREEN　3の1、2、3で説明ずみです。スプライト・パターンの大きさの

指定は、<スプライトサイズ> のところに行います。

スプライトサイズの指定は、0、1、2、3の値に指定します。

スプライトサイズに指定する0から3までの値は、それぞれ、つぎの図に示す大きさとなります。

値が1と3のスプライトサイズでは1マスが、1ビットを縦横2倍の拡大なので4ビットとなりますが、定義するときでは、1マスを1ビットと考えてください。

値	内　　　　　　　容
0	スプライトパターンを8×8ビットで構成します。
1	スプライトパターンを8×8ビットで構成し、縦横2倍に拡大した表示となります。
2	スプライトパターンを16×16ビットで構成します
3	スプライトパターンを16×16ビットで構成し、縦横2倍に拡大した表示となります。

さて、SCREENでスプライト面に表示するスプライト・パターンの大きさを指定しましたが、スプライト・パターンの大きさを指定しただけでは、スプライト・パターンをスプライト面に表示することはできません。

スプライト・パターンをスプライト面に表示するには、どんな形のスプライト・パターンであるか、その定義をしなければなりません。スプライト・パターンの定義は、SPRITE$で行います。SPRITE$については、229頁で説明しています。

また、SPRITE$で定義をしたからといって、スプライト・パターンが自動的にスプライト面に表示されるわけではありません。SPRITE$で定義したスプライト・パターンを、スプライト面に表示するにはPUT SPRITEで行います。PUT SPRITEについては、233頁で説明しています。

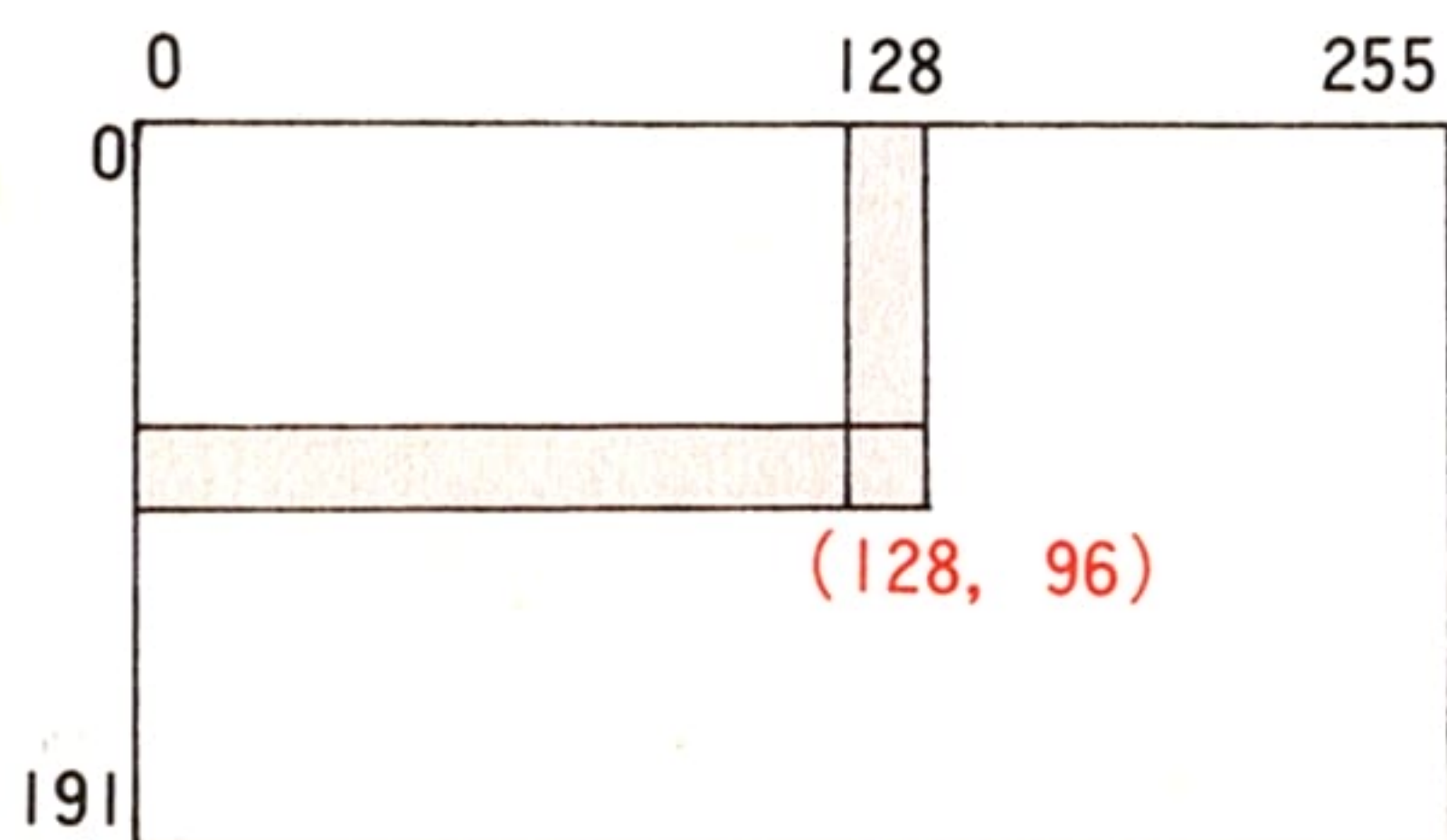
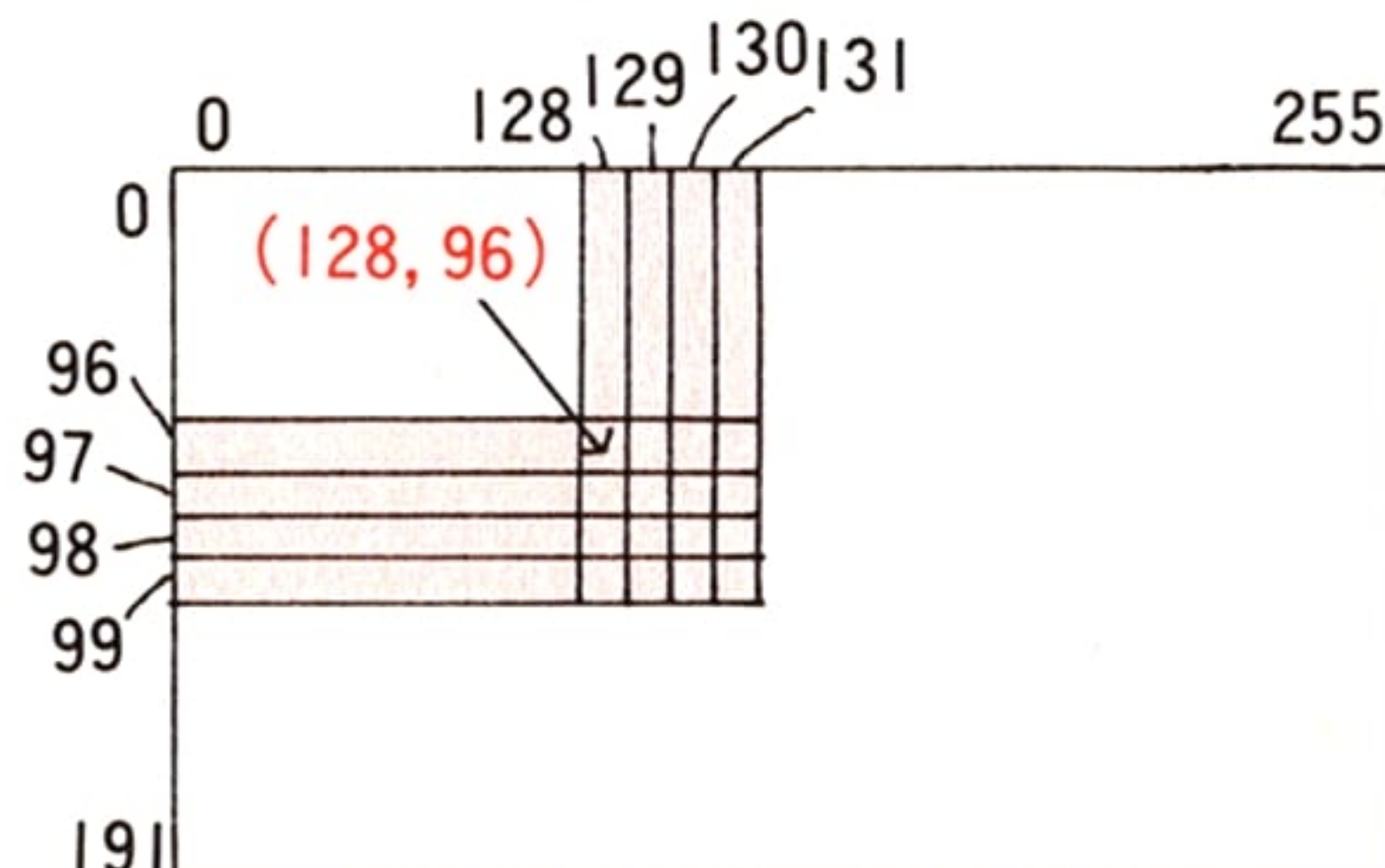
PSET

PSETは、ドットを表示する

PSET（ポイントセット）は画面にドットを表示します。PSETは、グラフィックモードで使う命令ですから、SCREENで2か3を指定しないと使うことができません。

PSET (X, Y), <カラーコード>

PSETのカッコのなかのXとYには、ドットを表示する位置を指定します。SCREEN 2あるいは3と指定すると、画面はグラフィック座標になりますから、Xにはグラフィック座標の列番号を指定し、Yには行番号を指定します。グラフィック座標の列番号は0から255まで、行番号は0から191までですから、その値のなかから、ドットを表示させる位置の値を選んで指定します。

たとえば、画面の真中にドットを表示させるとすると、列番号の真中は128ですから、128をXに与えます。行番号の真中は96ですから96をYに与えます。このようにすると、Xに与えた列番号128と、Yに与えた行番号96が交わる位置にドットが表示されます。

SCREEN 2の場合

SCREEN 3の場合

図に示したように、SCREEN 2の場合は、Xに指定した列座標128と、Yに指定した行座標96が交わった位置に、ドットがひとつ表示されます。

SCREEN 3の場合は、4×4の16ドットのブロック単位でドットを表示しますから、Xに指定した列番号128と、Yに指定した行番号96が交わった位置がブロックの左上のドットの表示位置となって、16個のドットが表示されます。

PSETの〈カラーコード〉のところには、0から15までのカラーコードのなかから、ひとつを選んで指定します。0から15までの値と色の関係は、172頁で説明しているCOLORのカラーコードとおなじです。したがって、PSETの〈カラーコード〉のところに8と指定すると、ドットが赤で表示されることになります。

つぎに示すのは、SCREEN 2で、画面の真中にドットを表示するプログラムです。

```
10 SCREEN 2
20 PSET(128,96),11
30 FOR T=1 TO 2000:NEXT T
40 END
```

つぎに示すのはSCREEN 3で、ドットを表示するプログラムです。

```
10 SCREEN 3
20 PSET(128,96),8
30 FOR T=1 TO 2000:NEXT T
40 END
```

▶グラフィックモードで実行するときの注意

前に示した2つのプログラムには、行番号30としてFOR～NEXTが入っています。このFOR～NEXTはウエイト・ループといいます。行番号30のFOR～NEXTは、FORの変数Tの初期値が1、最終値が2000となっていますから、変数Tの値が最終値の2000になるまで、FORとコロン「：」のあとのNEXTの間を行き来します。したがって、その間は行番号40のENDが実行されないので、ドットが画面に表示されていることになります。

このように、ウエイト・ループを入れている理由はSCREEN 2、SCREEN 3のグラフィックモードでは、プログラムの実行が終了すると、すぐにテキストモードに戻ってしまうからです。

グラフィックモードからテキストモードに戻るとき、グラフィックモードで画面に表示させたものを消してしまうので、何が表示されたのかわかりません。そこでウエイト・ループを入れて、ENDの実行を遅らせているのです。プログラムにENDを省略した場合も、おなじです。このことは、行番号30のウエイト・ループを取って実行させてみるとわかります。RUNするとすぐにテキストモードに戻ってしまうはずです。

グラフィックモードを使うときは、このように最後にウエイト・ループを置かなければならないということをおぼえておいてください。

なおつぎのように、行番号30にGOTOを置いて、繰り返えし行番号20のステートメントを実行させるような場合は、ウエイト・ループは必要ありません。

```
10 SCREEN 3
20 PSET(128,96),11
30 GOTO 20
```

PRESET

PRESETは、画面に表示したドットを消す

PRESET（ポイントリセット）は、PSETで画面に表示したドットを消すときに使います。PRESETでドットを消すには、PSET のXとYに指定した値とおなじ値を、PRESETのXとYに指定します。

PRESET　(X, Y)

たとえば、PSETでXに128、Yに96を指定して、ドットを表示したら、PRESETのXにも128、Yにも96を指定します。このようにすると、PSETで表示したドットがPRESETで消されます。つぎのプログラムを実行してみると、その様子がわかります。

```
10 COLOR ,14
20 SCREEN 3
30 PSET(128,96),4
40 FOR T=1 TO 5000:NEXT T
50 PRESET(128,96)
60 FOR T=1 TO 5000:NEXT T
70 GOTO 30
```

行番号10のCOLORでは、前景色の指定を省略して、背景色の指定だけをしています。表示するドットの色指定は、PSETの〈カラーコード〉のところにするからです。背景色は、カラーコード14で指定されていますから、灰色になります。このように前景色の指定を省略す

るときは、背景色のカラーコード14のまえに、コンマ「,」をつけるのを忘れないことです。

行番号20のSCREEN　3で、マルチカラーモードを指定しています。SCREEN　2にすると、1ドットしか表示されないので、表示が見にくいからです。そこで、SCREEN　3を指定して、4×4の16ドットを表示させるようにしています。

行番号30のPSETで、グラフィック座標の列番号を指定するXに128、行番号を指定するYに96を指定していますから、画面の真中あたりに4×4の16ドットを表示します。PSETの〈カラーコード〉のところには、カラーコード4を指定していますから、4×4の16ドットは、暗い青で表示されます。

行番号40のFOR～NEXTは、ウエイト・ループです。

このウエイト・ループは、行番号50のPRESETの実行を遅らせています。FORの変数Tが最終値の5000をこえるとウエイト・ループを抜け出して、行番号50のPRESETを実行します。

行番号50のPRESETのXとYの値は、行番号30のPSETのXとYの値とおなじです。したがって、行番号30のPSETで表示したドットが消されます。

行番号60のFOR～NEXTもウエイト・ループです。このウエイト・ループは、行番号70のGOTO　30の実行を遅らせています。GOTO　30の実行を遅らせているということは、行番号30がPSETですから、ドットの表示を遅らせているということです。

行番号40と60のウエイト・ループの働きで、4×4の16ドットは、ゆっくりと点滅を繰り返えします。もし、行番号40のウエイト・ループがないとすると、行番号50のPRESETがすぐ実行されるので、画面には何も表示されません。また、行番号60のウエイト・ループがないとすると、行番号30のPSETがすぐ実行されるので、ドットは表示されっぱなしになります。

LINE

LINEは、線を描いたり、箱を描く

LINE（ライン）は、SCREEN　2か3を指定して、グラフィックモードにして使う命令で、線を描いたり、箱を描いたりするときに使います。箱を描く方法には、2通りあります。ひとつは、線で箱の枠を描く方法です。もうひとつは、線で箱の枠を描いたあと、そのなかを塗りつぶす方法です。

まず、LINEで線を描くことから説明することにします。

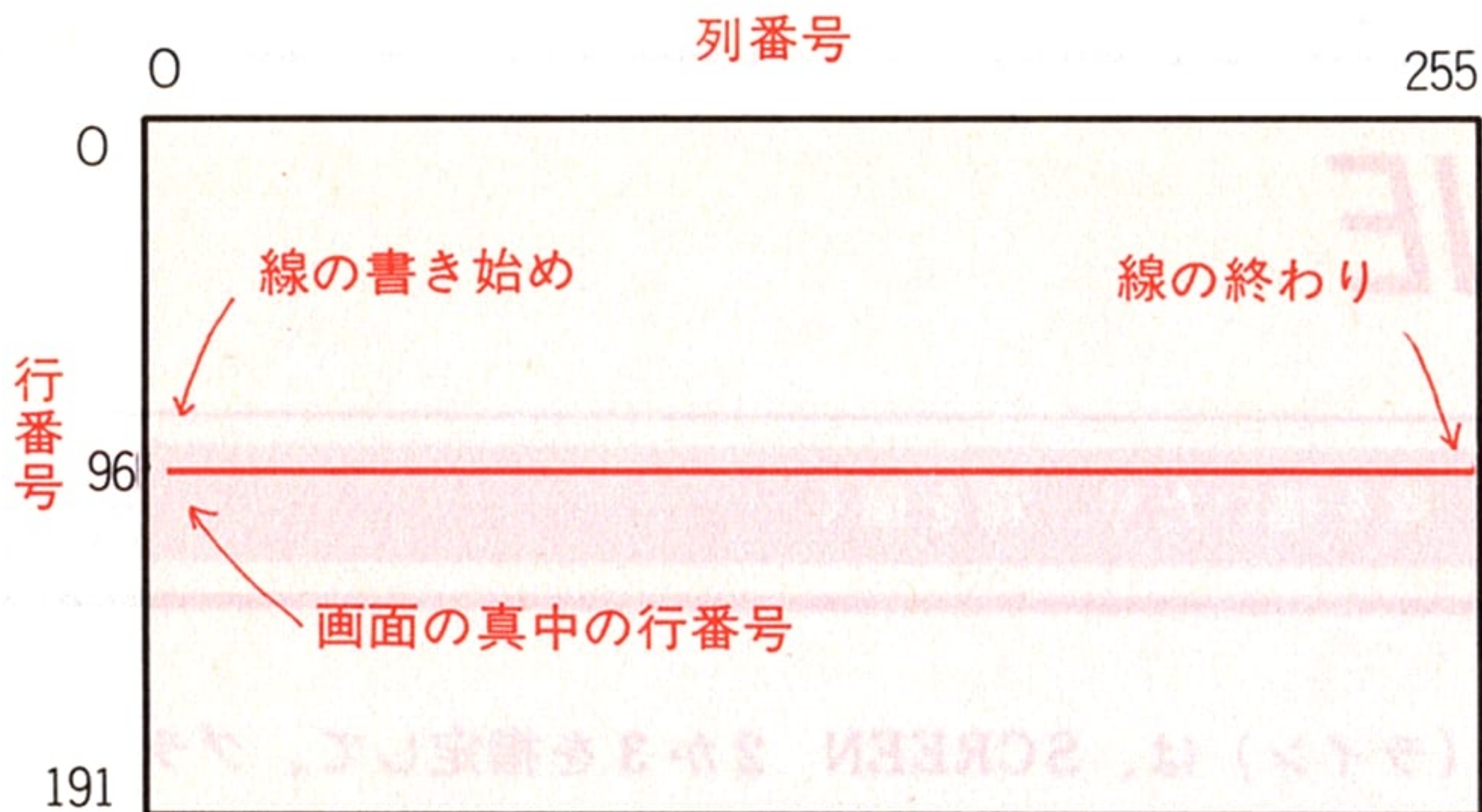
▶線を描く

LINEで線を描くには、線を描く最初の位置と最後の位置を指定します。

LINE　(X1,　Y1)－(X2,　Y2),〈カラーコード〉

LINEの最初のカッコのなかのX1とY1には、線を描く最初の位置を指定します。つぎのカッコのなかのX2とY2には、線の最後の位置を指定します。

(X1,　Y1)　　　　(X2,　Y2)

SCREEN　2と3はグラフィックモードですから、X1とX2には、グラフィック座標の列番号を指定します。Y1とY2には、行番号を指定します。たとえば、画面の真中に、画面の左端から右端まで線を引くとします。画面の真中の位置は、つぎの図に示すように行番号96です。画面の左端は列番号0、右端は255です。

したがって、X 1 に画面の左端の列番号 0 、Y 1 に行番号96を与えます。X 2 には画面の右端の列番号255、Y 2には行番号96を与えます。LINEの〈カラーコード〉のところには、線を黄色で描くとすれば、11を指定します。11は黄色のカラーコードです。これをまとめると、つぎのようになります。

LINE （0，96）－（255，96），11

これだけでは実行できませんから、プログラムにまとめて示します。

```
10 COLOR ,8
20 SCREEN 2
30 LINE(0,96)-(255,96),11
40 FOR T=1 TO 5000:NEXT T
50 END
```

行番号10のCOLOR ，8 は、背景色の指定です。カラーコード 8 は赤ですから、バックは赤になります。行番号20のSCREEN 2 の指定は高解像度グラフィックモードですから、1 ドットずつで線を描くことになります。行番号30のLINEで画面の真中を、画面の左端から右端まで、黄色で線を描きます。行番号40のFOR～NEXTはウエイト・ループ（212頁参照）です。

このプログラムを実行させせると、細い線が描かれますが、このような細い線ではなくて、太い線を描きたいときは、行番号20のSCREEN 2を、

```
20 SCREEN 3
```

に変更すれば、太い線を描くことができます。

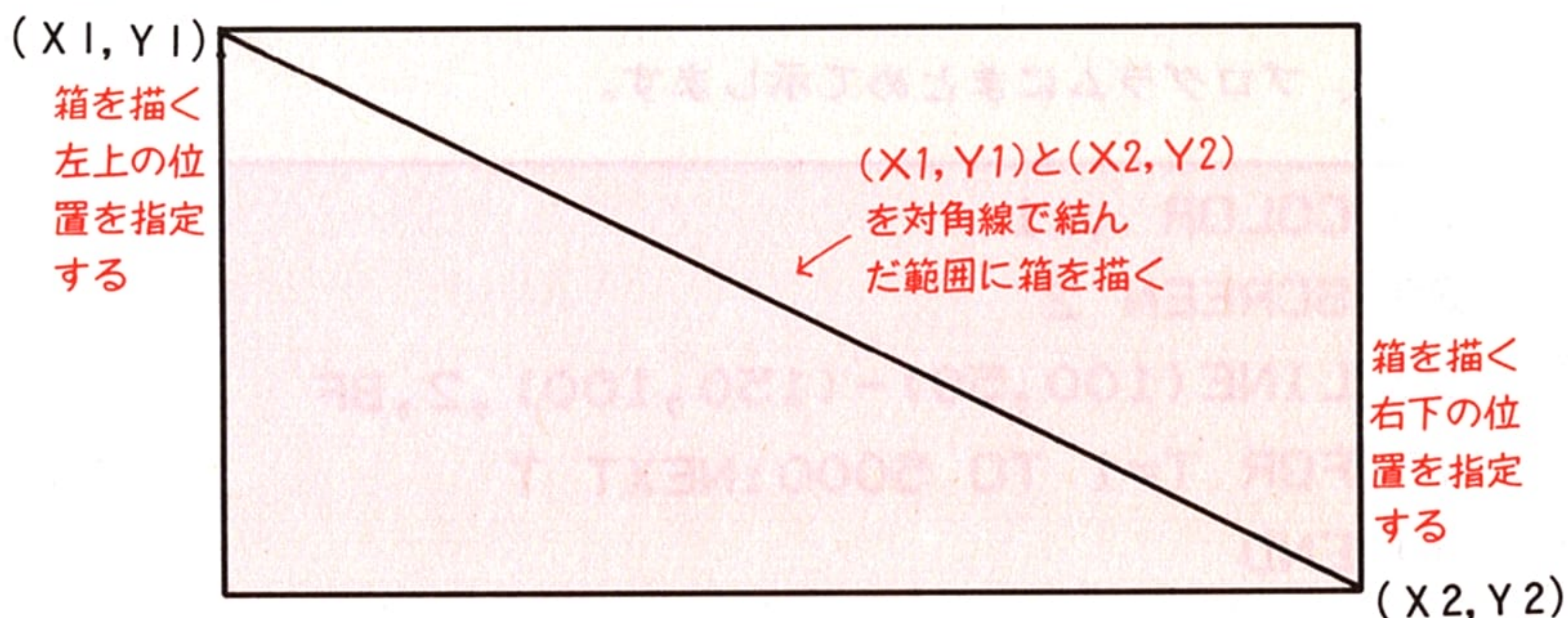
線の描き方は、このようにLINEのカッコのなかに、線の描きはじめの位置と終わりの位置を指定すればよいのです。

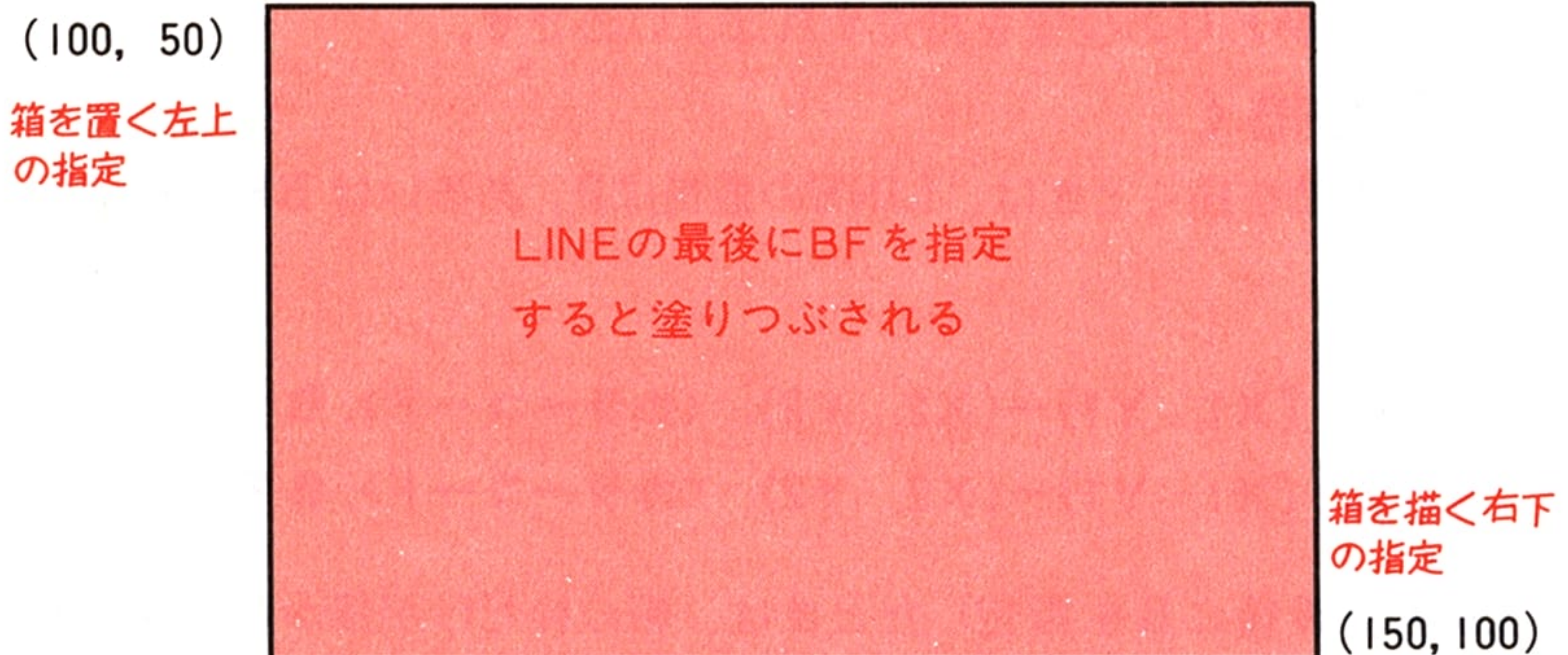
▶箱を描く

LINEで箱を描くときは、LINEの最後にB、あるいはBFを指定します。

LINE (X1, Y1)−(X2, Y2), 〈カラーコード〉, B
LINE (X1, Y1)−(X2, Y2), 〈カラーコード〉, BF

LINEの最後にBを指定したときは、箱の枠だけが描かれます。BFを指定したときは、箱の枠が描かれて、そのなかが塗りつぶされます。箱のなかを塗りつぶす色は、LINEの〈カラーコード〉に指定した色によってです。つまり、枠を描く色とおなじ色で塗りつぶされるわけです。

さて、LINEの（X１, Y１）と（X２, Y２）の指定です。

（X 1， Y 1）には、まえの頁の図に示したように、箱を描く左上の位置を指定します。（X 2， Y 2）には、箱を描く右下の位置を指定します。このように指定すると、（X 1， Y 1）と（X 2， Y 2）を対角線で結んだ範囲に箱を描きます。

たとえば、グラフィック座標の列番号100、行番号50の位置から、列番号150、行番号100の位置にかけて箱を描くときは、X 1 に100、Y 1 に50、X 2 に150、Y 2 に100を指定します。

この箱を緑で描くとすれば、〈カラーコード〉に 2 を指定します。この場合、LINEの最後をBFとすると、カラーコードに指定した緑色で枠を描き、そのなかを緑色で塗りつぶすことになります。

LINE (100, 50)−(150, 100), 2, BF

このステートメントだけでは実行できませんから、つぎに実行できるように、プログラムにまとめて示します。

```
10 COLOR ,11
20 SCREEN 2
30 LINE(100,50)-(150,100),2,BF
40 FOR T=1 TO 5000:NEXT T
50 END
```

CIRCLE

CIRCLEは、円や楕円を描くときに使う

CIRCLEは、円や楕円を描くときに使います。CIRCLEは、SCREEN　2、あるいはSCREEN　3を指定して使いますが、SCREEN　3を指定すると、ドットが16ドットずつで表示されるので、描かれた円や楕円が粗いものになります。したがってここでは、SCREEN　2の指定で説明することにします。

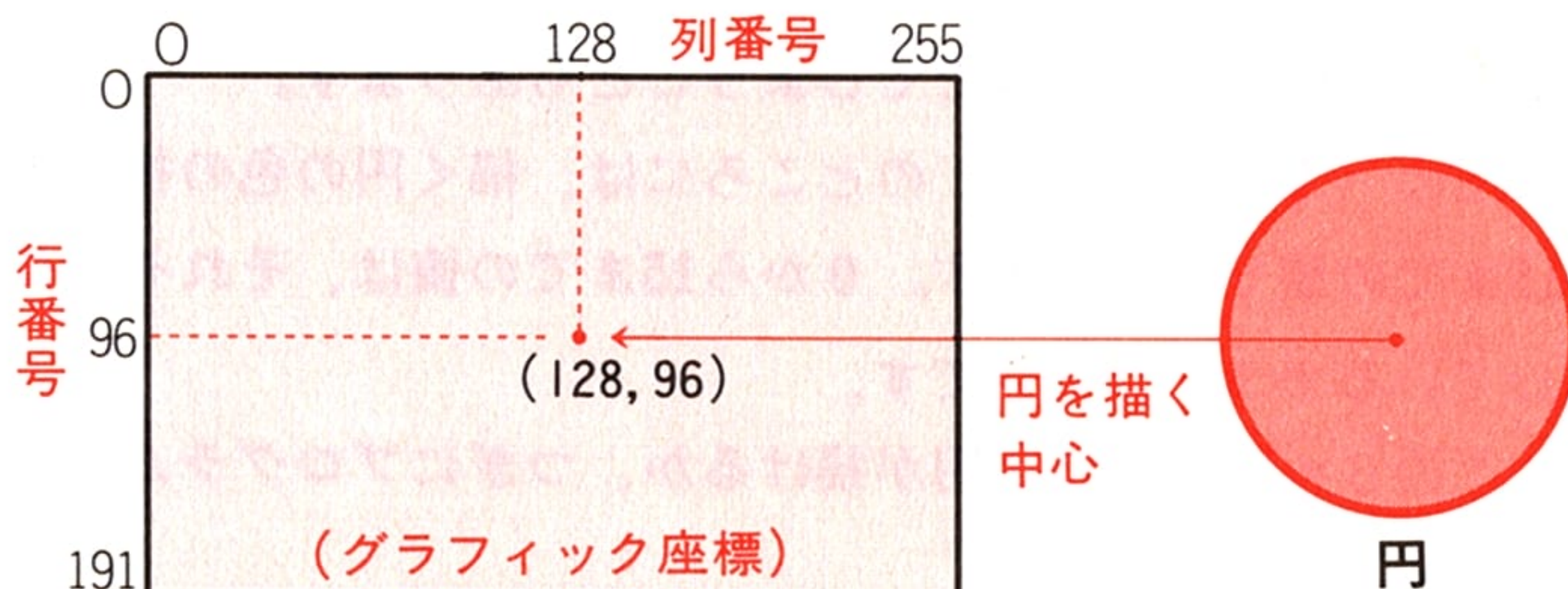
CIRCLEは、つぎのように複雑な形をしています。

CIRCLE（X，Y），<半径>，<カラーコード>，
<開始角度>，<終了角度>，<比率>

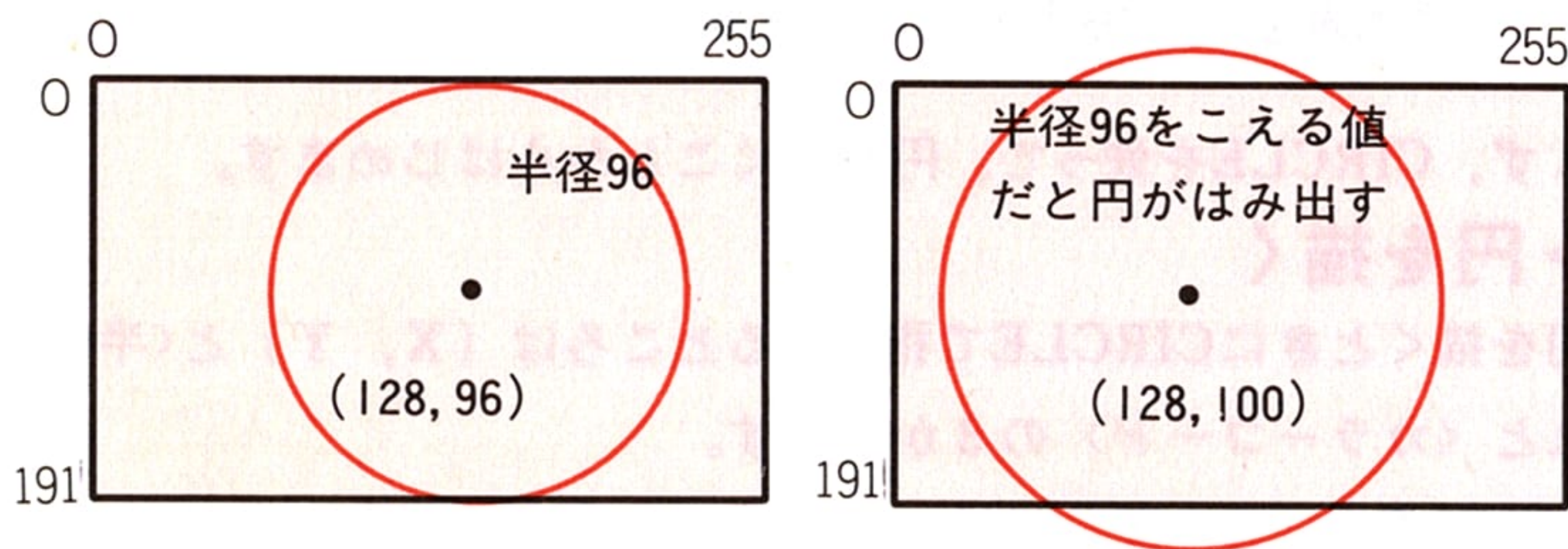
まず、CIRCLEを使って、円を描くことからはじめます。

▶円を描く

円を描くときにCIRCLEで指定するところは（X，Y）と<半径>、それと<カラーコード>の3か所です。

CIRCLEの（X，Y）には、円を描く中心点を指定します。中心点の指定は、グラフィック座標で行います。

グラフィック座標はまえの図に示したように、列番号が0から255まで、行番号が0から191までです。たとえば、画面の真中に円を描くとすれば、列番号の真中は128、行番号の真中は96ですから、列番号を指定するXに128、行番号を指定するYに96を与えます。するとまえの図に示したように、列番号128と行番号96が交わった位置が円を描く中心点となって、円が描かれます。

CIRCLEの〈半径〉のところには、描く円の半径を指定しますが、与える値はドット数で与えます。つまり、半径何ドットの大きさの円を描くというわけです。

画面に表示できる横のドット数は256、縦のドット数は192です。円を描く中心点を、Xが128、Yが96というときは、半径96が指定できる最大の値となります。半径に96以上の値を指定すると、描かれる円が画面からはみ出ることになります。

ただし、これはあくまで理論上のことで、半径を96と指定しても、円の上から下が画面からはみでてしまうこともあります。

CIRCLEの〈カラーコード〉のところには、描く円の色の指定を、0から15までの値で指定します。0から15までの値は、それぞれの色と対応しているカラーコードです。

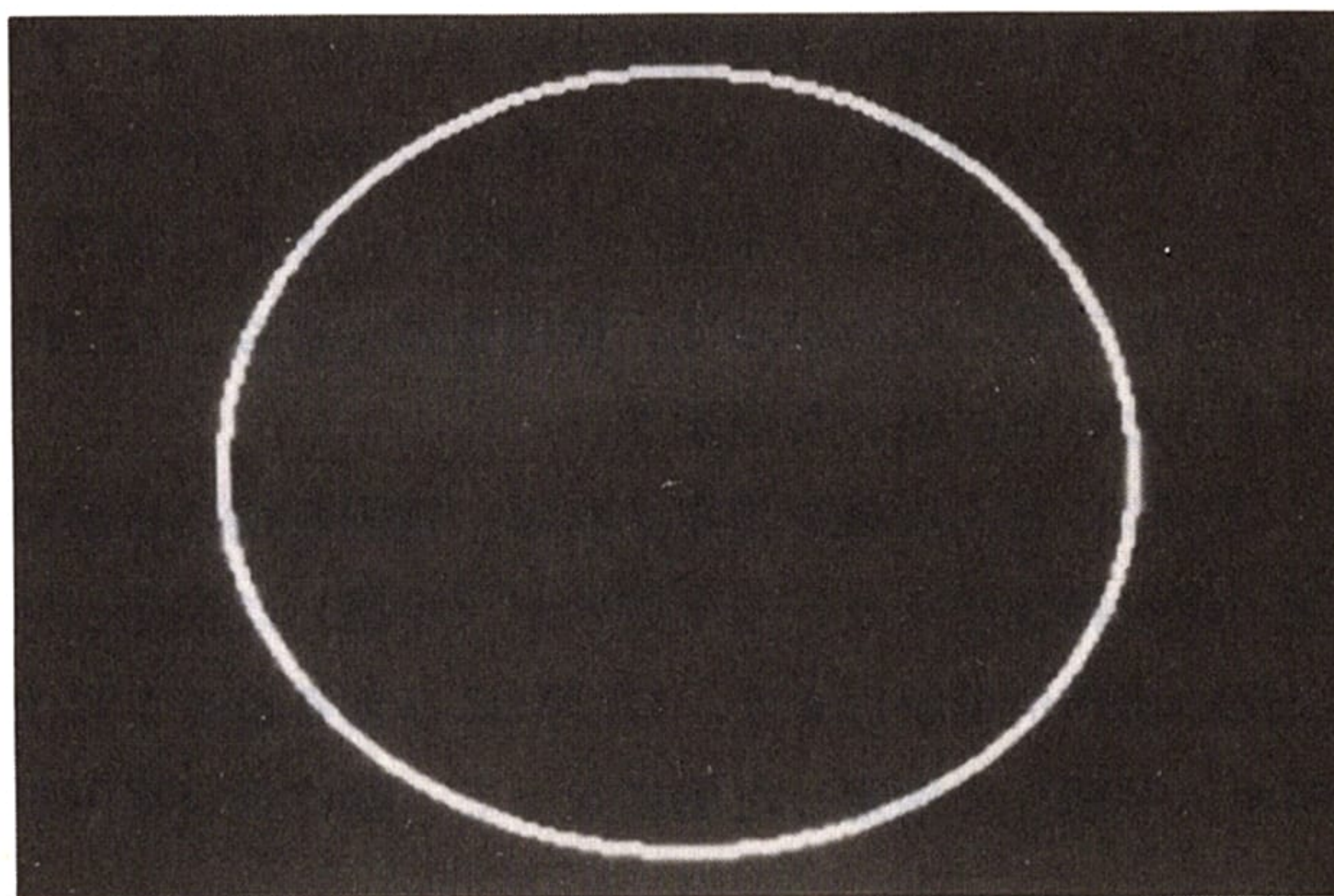
では、この3か所の指定で円が描けるか、つぎにプログラムを示します。

```
10 COLOR ,10
20 SCREEN 2
30 CIRCLE(128,96),80,12
40 FOR T=1 TO 5000:NEXT T
50 END
```

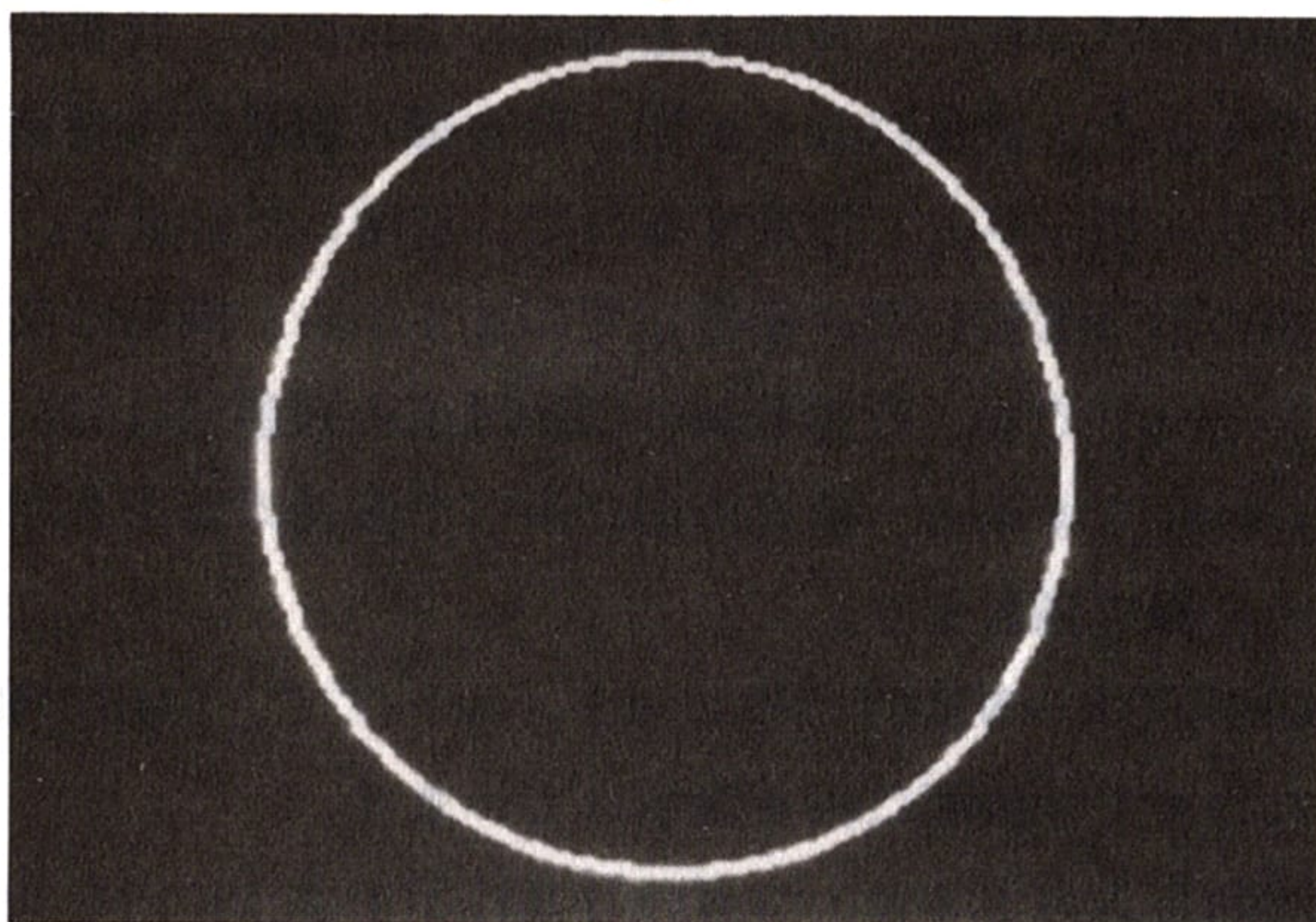
行番号10のCOLORで、背景色を指定しています。背景色はカラーコード10ですから、黄色になります。円を描く前景色の指定は、CIRCLEの〈カラーコード〉で指定するので、省略しています。

行番号20のSCREENで2を指定していますから、画面は高解像度グラフィックモードです。行番号30のCIRCLEの列番号は128、行番号は96、〈半径〉は80ですから、前頁の図に示した画面の真中を中心にして、半径80の円を緑で描きます。

行番号40のFOR～NEXTはウエイト・ループで、FORの変数Tの値が5000になるまで、画面に円を表示しています。

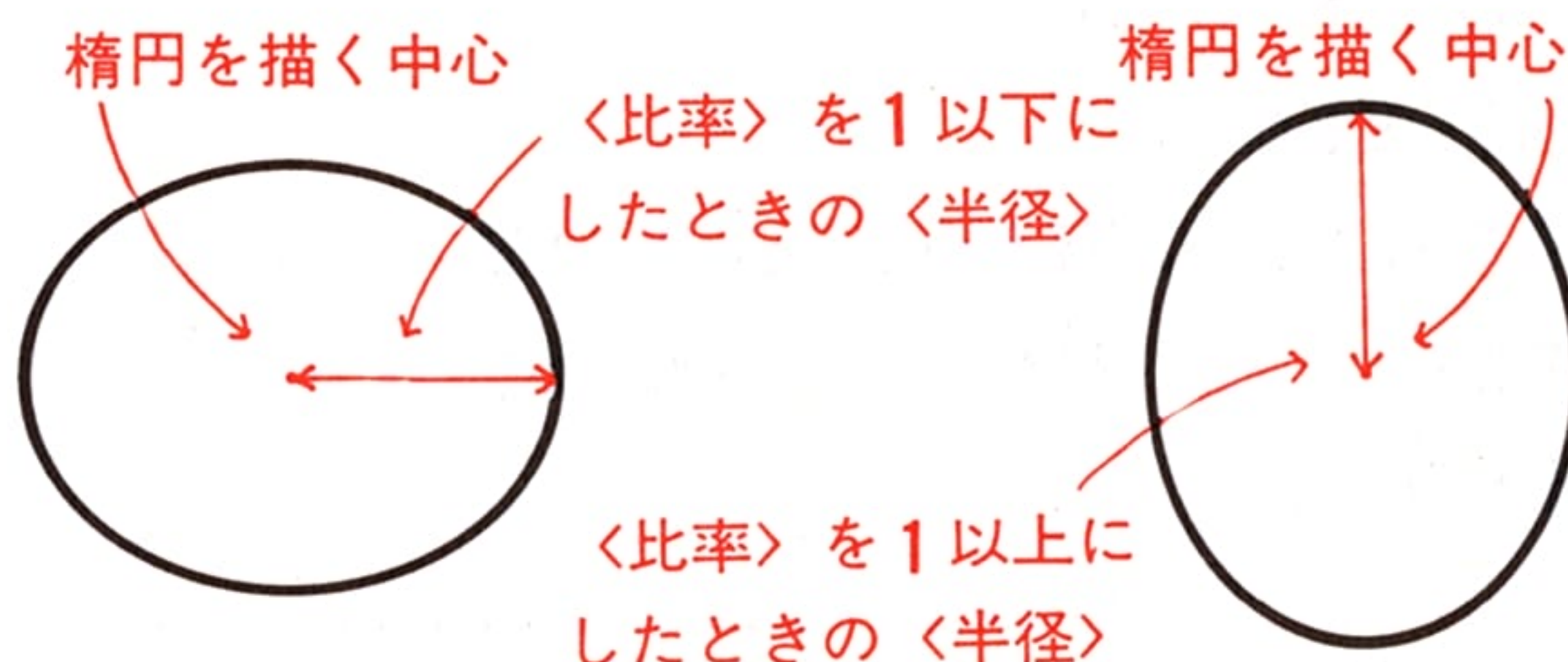
さて[R][U][N]を入力すると、つぎの写真に示すように円を表示しますが、どう見ても円とはいえません。これでは楕円です。

このようになるのは、画面構成の関係です。円にちかづけるには、CIRCLEの〈比率〉のところに値を与えます。〈比率〉については楕円を描くところで説明しますので、ここではつぎのように、〈比率〉のところに1.2を与えてください。

```
30 CIRCLE(128,96),80,12,,,1.2
```

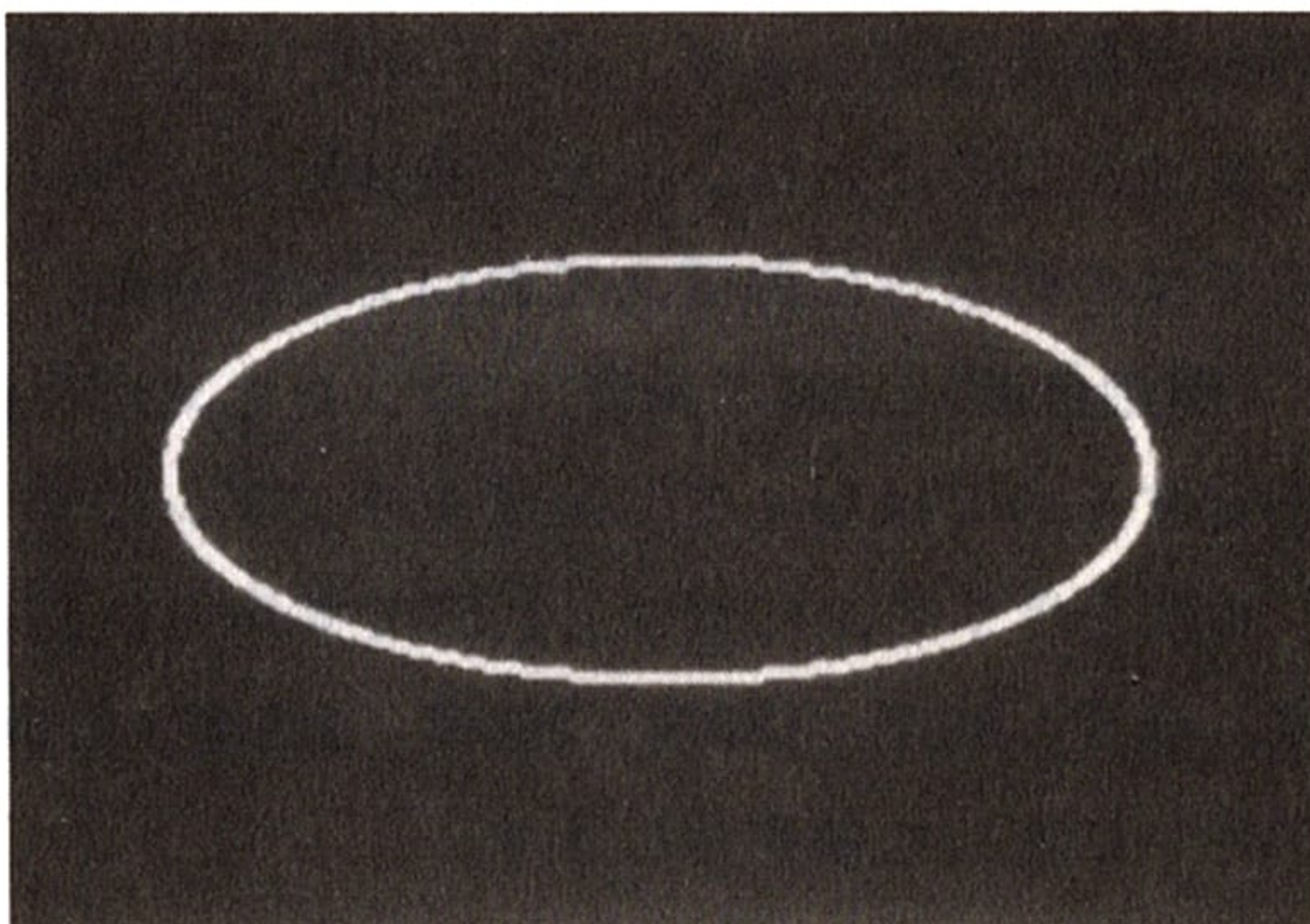
このように実行させると、円になります。

▶楕円を描く

楕円を描くときにCIRCLEで指定するところは、(X, Y)と〈カラーコード〉、それに〈半径〉と〈比率〉のところの4か所です。(X, Y)と〈カラーコード〉、それと〈半径〉の指定は、さきに説明した円を描く場合とおなじです。したがって、楕円を描くには、〈比率〉に値を指定すればよいということになります。

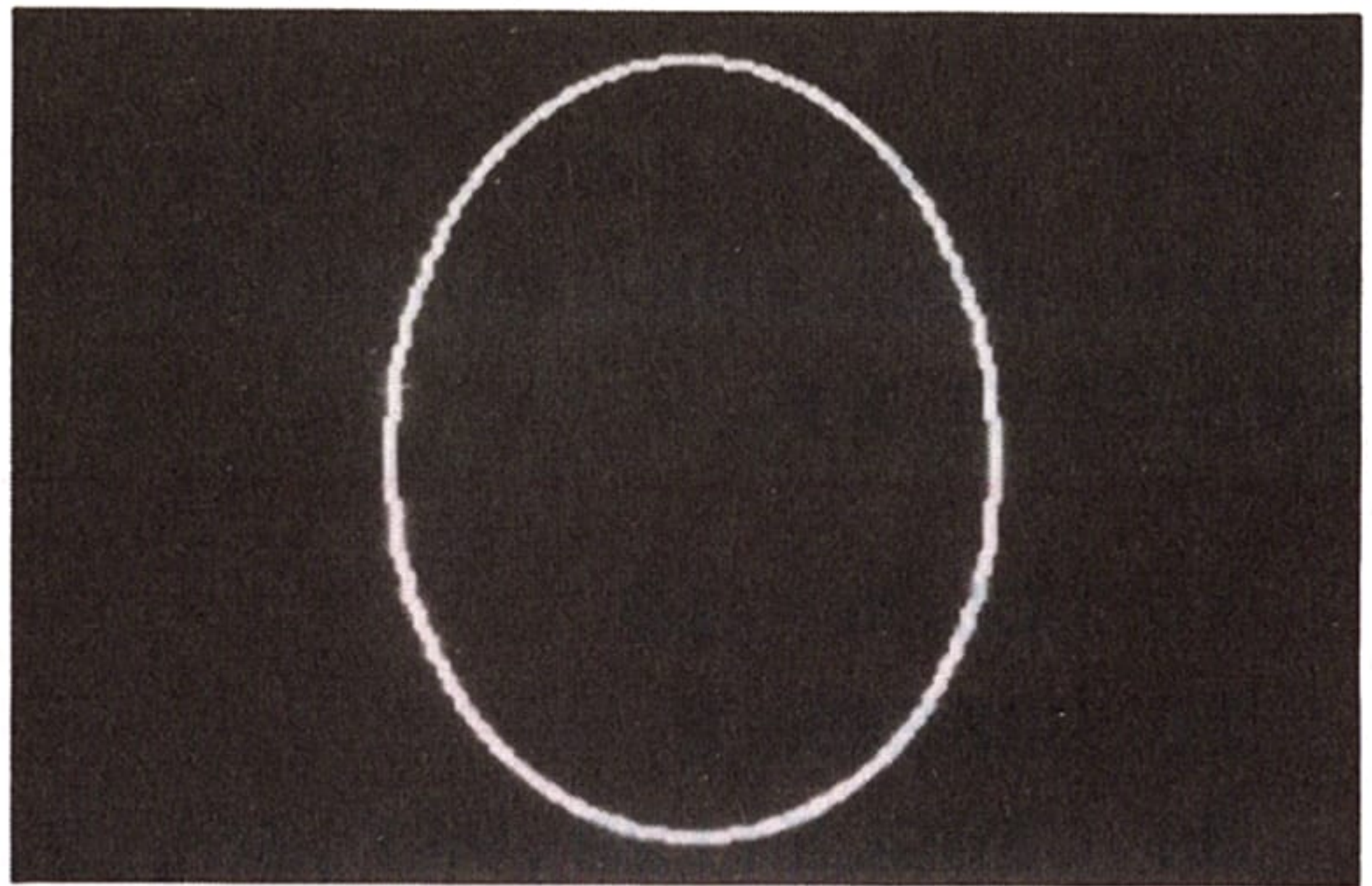
では、〈比率〉にどんな値を指定すれば、楕円を描けるかです。〈比率〉に1以下の値を指定すると、水平方向が〈半径〉に指定した値になって、横に長い楕円が描かれます。〈比率〉に与える値を1以上にすると、垂直方向が〈半径〉に指定した値になって、縦に長い楕円が描かれます。

比率＝1以下にすると、横に長い楕円が描かれる
比率＝1以上にすると、縦に長い楕円が描かれる

つぎに、〈比率〉のところに、0.5を指定したプログラムを示します。そのほかの指定は、219頁に示した円を描くプログラムの指定とおなじにすることにします。

```
10 COLOR ,10
20 SCREEN 2
30 CIRCLE(128,96),80,12,,,.5
40 FOR T=1 TO 5000:NEXT T
50 END
```

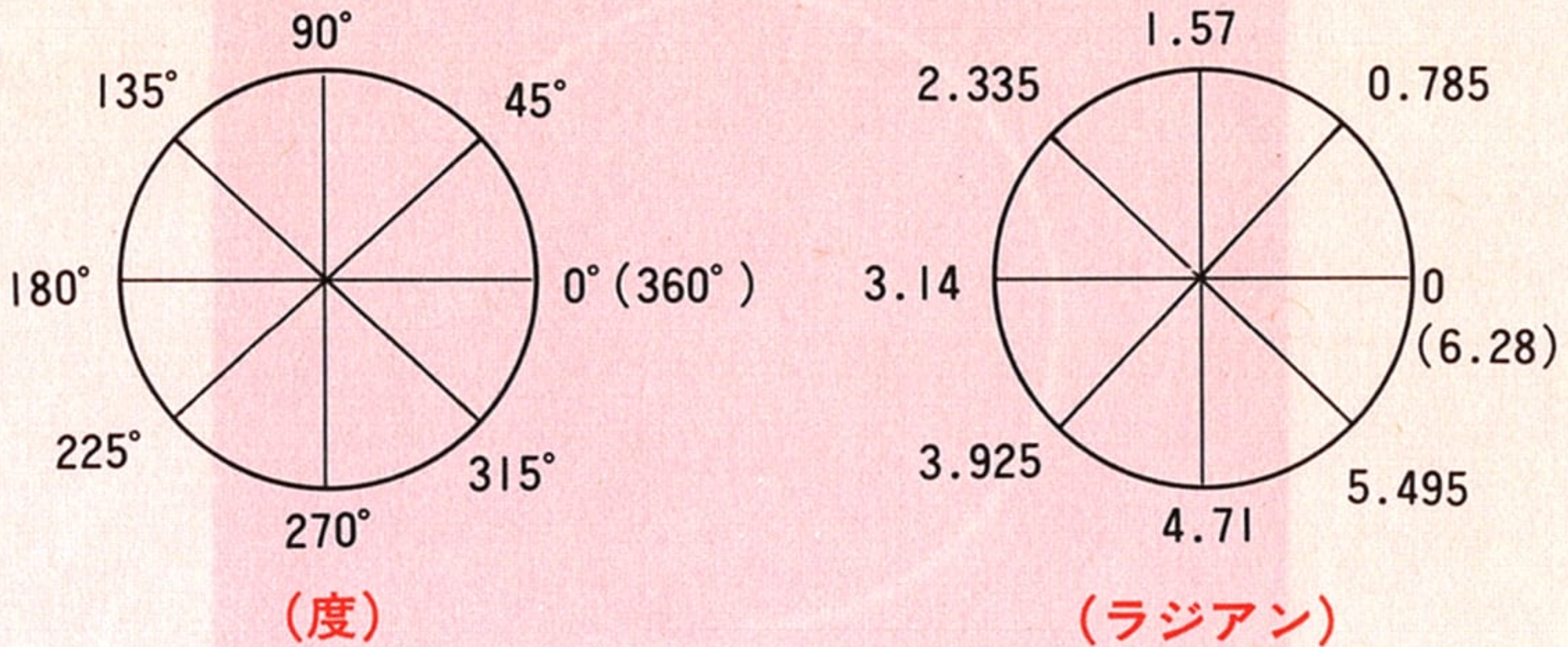
このプログラムを実行すると、〈比率〉が1より小さい0.5ですから、水平方向が〈半径〉に指定した80となって、右の写真に示すように横に長い楕円が描かれます。

つぎに、〈比率〉のところに、1.5を指定したプログラムを示します。

```
10 COLOR ,10
20 SCREEN 2
30 CIRCLE(128,96),80,12,,,1.5
40 FOR T=1 TO 5000:NEXT T
50 END
```

このプログラムを実行させると、〈比率〉が1より大きい1.5となっていますから、垂直方向が〈半径〉に指定した80になって、右の写真に示すように、縦に長い楕円が描かれます。

▶〈開始角度〉と〈終了角度〉の指定

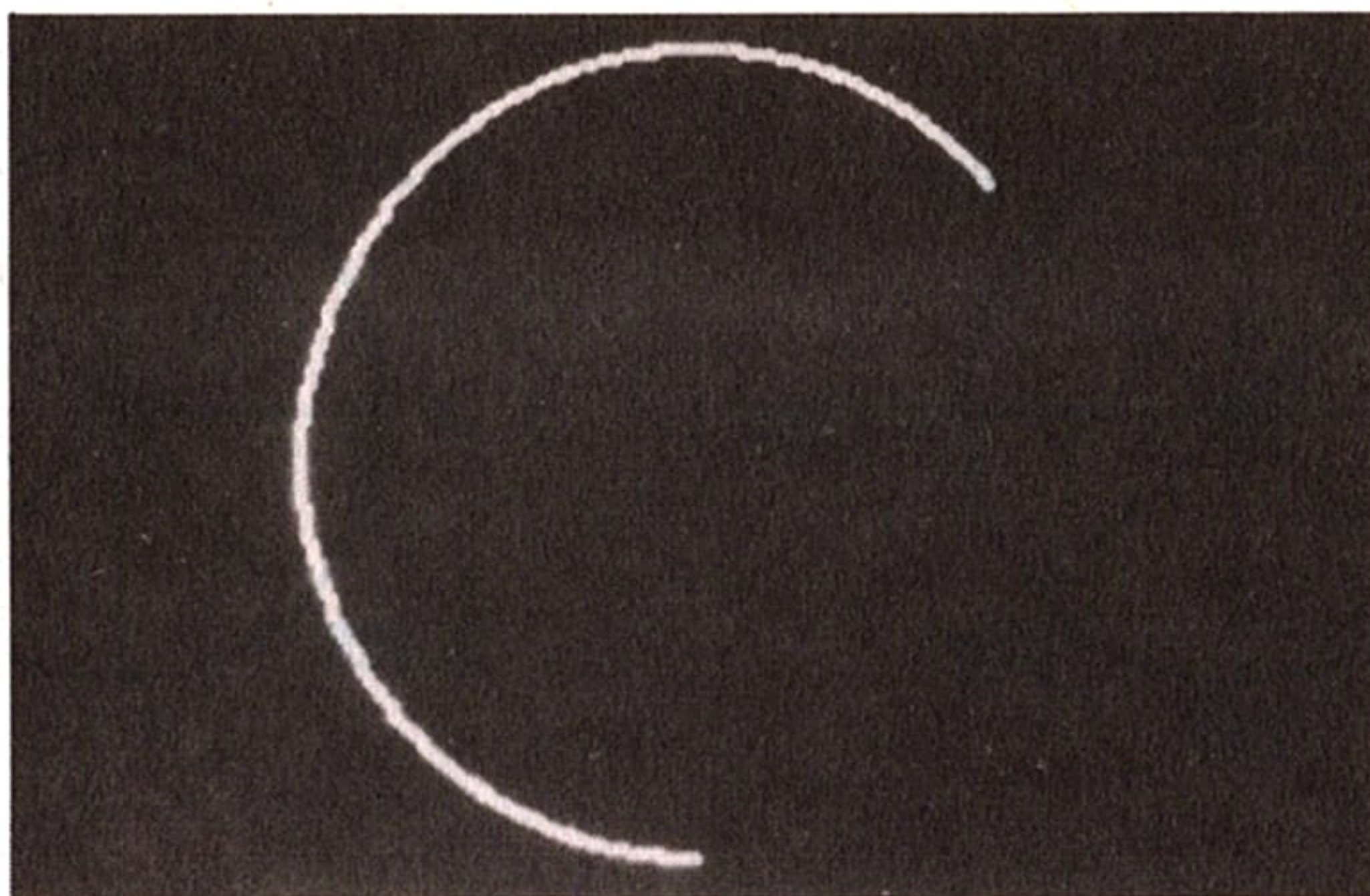
さて、残るところは、CIRCLEの〈開始角度〉と〈終了角度〉の指定です。〈開始角度〉には、円や楕円、それに円弧などの線を描きはじめる位置を指定します。〈終了角度〉には、線を描き終わる位置を指定します。このことは、あとで具体的にプログラムで示しますから、ここではこのようにおぼえておいてください。

〈開始角度〉と〈終了角度〉には、たとえば、90度というように角度で指定するのではなくて、90度をラジアンに直して指定します。

90度をラジアンに直すと1.57ですから1.57と指定します。つぎに0度、45度、90度、135度、180度、225度、270度、315度、360度を、ラジアンに直した値を示します。

では、CIRCLEの〈開始角度〉と〈終了角度〉を指定して、円弧を描いてみることにします。この場合は、円弧を45度の位置から描きはじめて、270度の位置で終わることにします。したがって、CIRCLEの〈開始角度〉のところに45度をラジアンに直した 0.785 を指定します。〈終了角度〉のところは、270度をラジアンに直した4.71を指定します。CIRCLEのそのほかの指定は 219 頁で描いた円の指定とおなじにすることにします。

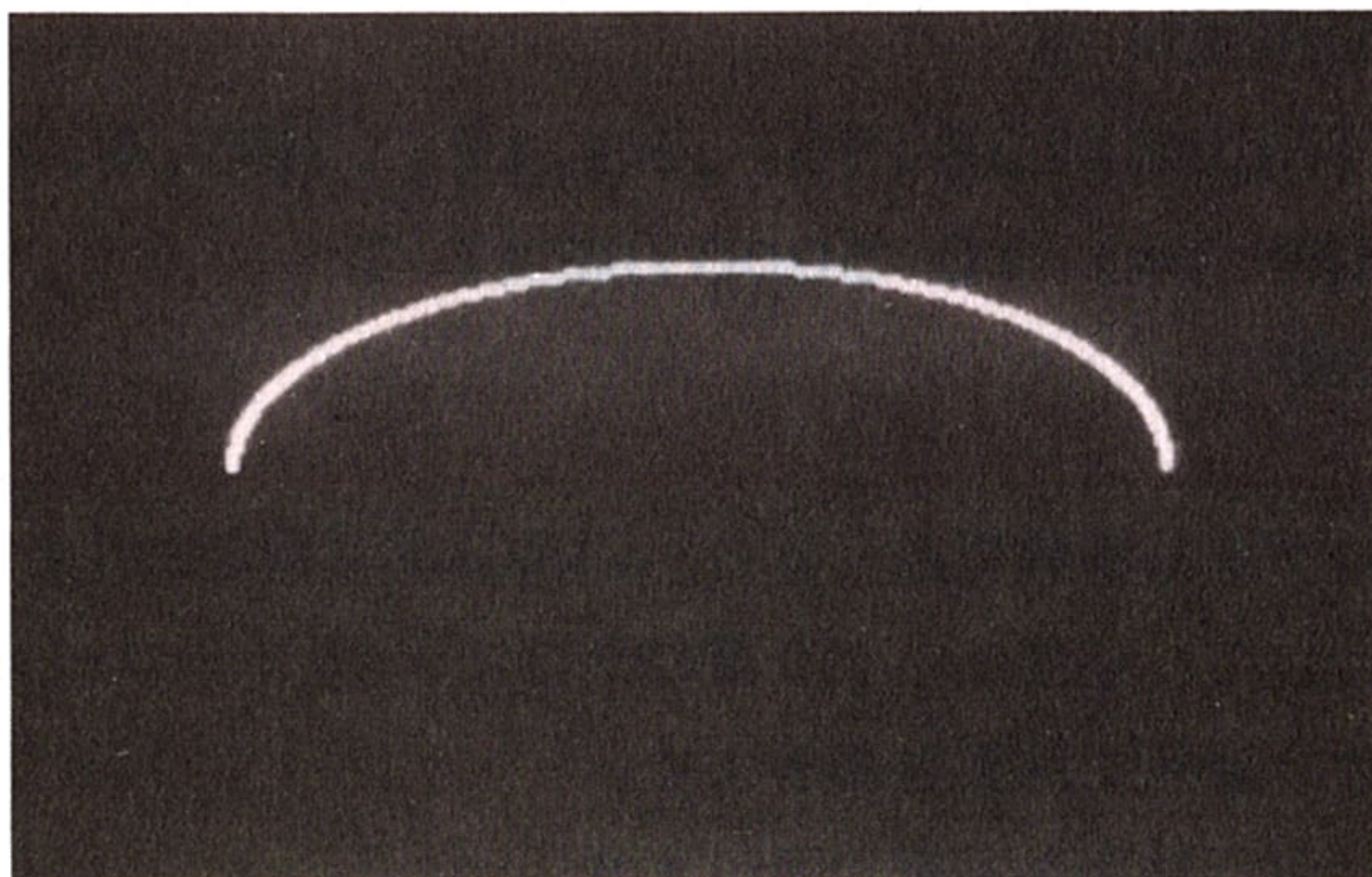
```
10 COLOR ,10
20 SCREEN 2
30 CIRCLE(128,96),80,12,.785,4.71,1.2
40 FOR T=1 TO 5000:NEXT T
50 END
```

このプログラムを実行させると、CIRCLEに指定された(128,96)を中心点にして、45度から 270 度にかけて、〈カラーコード〉に指定した12、つまり緑色で円弧を描きます。

つぎに、実行結果を写真で示します。

楕円の円弧を描くときは、CIRCLEの〈比率〉のところに1以下、あるいは1以上の値を指定します。〈比率〉に1以下の値を指定したときは、横に長い楕円の円弧を描くことができます。〈比率〉に1以上の値を指定したときは、縦に長い楕円の円弧を描くことができます。

円弧を描いたさきのプログラムの行番号30を、つぎのように変更して実行させると、下の実行結果の写真に示すように、横に長い楕円の円弧、縦に長い楕円の円弧を描きます。

```
30 CIRCLE(128,96),80,12,0,3.14,.5
```

```
30 CIRCLE(128,96),80,12,1.57,4.71,1.5
```

PAINT

PAINTは、円などの図形のなかを塗る

LINEを使って箱を描いたとき、LINEの終わりにBFという記号をつけると、箱のなかを塗ることができます。つまり四角形を描いて、そのなかを塗るのとおなじです。円や三角形などの図形を描いたときは、LINEのようなわけにはいきません。

円や三角形などの図形を描いてそのなかを塗るときは、PAINT（ペイント）を使って塗ることになります。

PAINTは、SCREEN 2を指定したときと、SCREEN 3を指定したときでは、その働きが異なります。

まず、わかりやすいように、PAINTの形を示します。

PAINT（X，Y），〈領域色〉，〈境界色〉

PAINTのカッコのなかのXとYには、図形のなかを色で塗り始める位置を指定します。Xには、グラフィック座標の列番号を指定します。Yには、行番号を指定します。このことは、あとで具体的に説明します。

〈領域色〉には、図形のなかを塗る色をカラーコードで指定します。たとえば、図形のなかを黄色で塗ろうとするときは〈領域色〉のところに黄色のカラーコード11を指定します。すると、図形のなかが黄色で塗られることになります。

〈境界色〉には、図形を描いた色をカラーコードで指定します。たとえば、図形を赤で描いたとしたら、〈境界色〉のところに赤のカラー

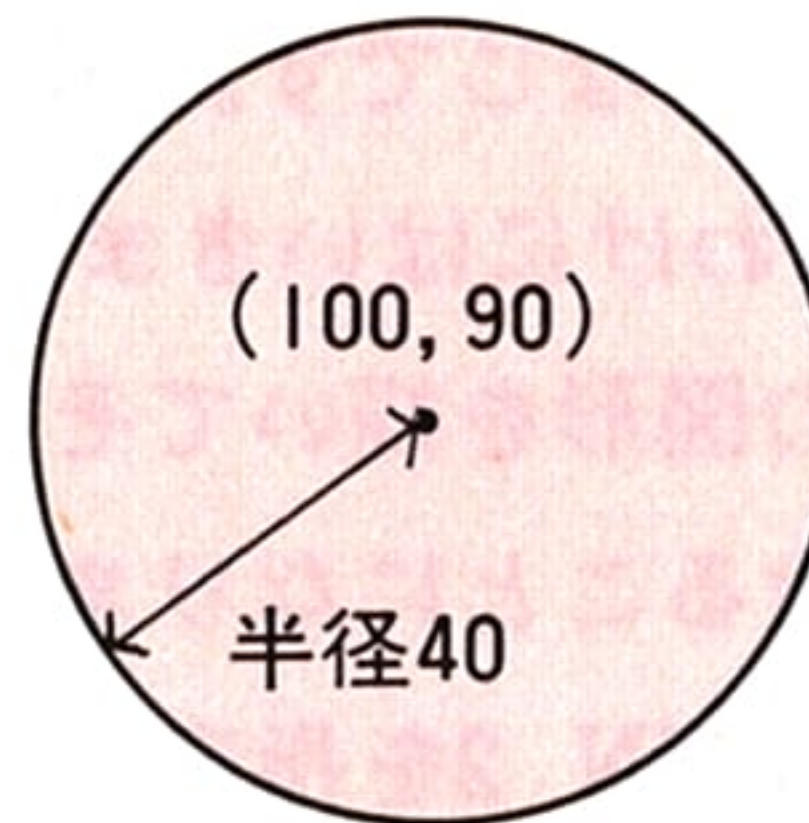
コード8を指定します。すると〈境界色〉に指定した赤で描かれた図形のなかが〈領域色〉に指定した色で塗られることになります。

▶SCREEN 2を指定したときのPAINT

まず、SCREEN 2を指定したときのPAINTから説明します。ここでは具体的に説明する関係から、円を描いて、その円のなかを塗る例を取りあげます。

円は、グラフィック座標の列番号100、行番号90の位置を中心点にして、半径40で描くことにします。円を描く色は、緑色にします。

CIRCLE (100,90), 40, 2, , , 1.2

これで、みぎの図に示すように円を描くことができます。CIRCLEの最後にある1.2は、円にするための比率です。このことについては、220頁を参照してください。さて緑色で描いた円のなかを塗ることにします。まず、PAINTのカッコのなかのXとYに、円のなかを色で塗りはじめる位置を指定します。円は、列番号100と行番号90を中心点にして描かれていますから、PAINTのカッコのなかのXとYに、その値を与えます。

列番号100、行番号90を中心として、半径40の円を描く

PAINT (100,90)

このように指定すると、列番号100と行番号90から、うえに向かって色を塗りはじめ、つぎに下に向かって色を塗っていきます。

色を塗りはじめる列番号と行番号は、円の中心でなければならないというわけではありません。この場合は、つぎのように、

PAINT (80,70)

としても、円のなかを塗ることができます。つまり、円のなかの列番号、行番号をPAINTのカッコのなかのXとYに指定すればよいとい

うことです。

　さてつぎは、〈領域色〉と〈境界色〉の指定です。円は緑色で描いていますから〈境界色〉の指定は２、緑色で描いた円のなかは明るい緑で塗るとして、明るい緑のカラーコード３を〈領域色〉に指定したとします。

　　PAINT（100, 90），３，２

　ところが、このようにするとエラーになって、円のなかを塗ることができません。画面全体が塗られてしまいます。つぎのプログラムを実行させてみるとわかります。

```
10 COLOR  ,11
20 SCREEN 2
30 CIRCLE(100,90),40,2,,,1.2
40 PAINT(100,90),3,2
50 FOR T=1 TO 5000:NEXT T
60 END
```

　なぜこのようなことになるのかというと、SCREEN　2のときは、円を描いた色とおなじ色でしか、円のなかを塗ることができないようになっているからです。

　この場合は、緑色のカラーコード２で円を描きましたから、円のなかを塗る色も、カラーコード２の緑色となります。したがって〈領域色〉のところに、緑色のカラーコード２を指定します。緑色にもいろいろあって、カラーコードもそれぞれ違いますから、カラーコードを間違えないことです。〈境界色〉のところの指定は必要ありません。したがって、SCREEN　２を指定したときのPAINTは、つぎのようになります。

```
40 PAINT(100,90),2
```

さきに示したプログラムの行番号40を、このステートメントに書きかえて実行すると、円のなかが緑色に塗られます。

▶SCREEN 3を指定したときのPAINT

つぎは、SCREEN 3を指定したときです。SCREEN 3を指定したときのPAINTの働きは、SCREEN 2と違います。つまり<領域色>と<境界色>の指定ができるようになります。ということは、ある色で描いた図形のなかを、違った色で塗ることができるということです。

ここでも説明の都合上、まえの場合とおなじように、グラフィック座標の列番号100、行番号90を中心として、半径60の円を描いて円のなかを塗ることにします。円を描く色は青にします。

CIRCLE (100, 90), 60, 4, , , 1.2

さて、青で描いた円のなかは、シアンで塗ることにします。円のなかをシアンで塗りはじめる位置は、円を描いた中心からにしますからPAINTのXとYには列番号100と90を与えます。

円は青で描かれていますから、<境界色>には青のカラーコード4を指定します。円のなかはシアンで塗りますから、<領域色>にはシアンのカラーコード7を指定します。

PAINT (100, 90), 7, 4

これで、青で描いた円のなかをシアンで塗ることができます。つぎにプログラムをまとめて示します。

```
10 COLOR  ,11
20 SCREEN 3
30 CIRCLE(100,90),60,4,,,1.2
40 PAINT(100,90),7,4
50 FOR T=1 TO 5000:NEXT T
60 END
```

SPRITE$とPUT SPRITE

**SPRITE$でキャラクタを記憶させて、PUT SPRITEで
スプライト面に、そのキャラクタを表示させる**

202頁のSCREENのところでSCREENの〈画面モード〉のところを、1、2、3と指定すると、キャラクタを表示するスプライト機能が使えると説明しました。また、SCREENの〈スプライトサイズ〉には、0、1、2、3の値を指定することができて、それぞれの値は、スプライト面に表示するキャラクタの大きさを指定すると説明しました。

SPRITE$は、SCREENで指定した大きさのキャラクタを記憶させるために使います。

SPRITE$ 〈スプライトパターン番号〉

SPRITE$の〈スプライトパターン番号〉には、スプライト面に表示するスプライトパターン、つまりキャラクタの形や大きさを記憶しておく番号を指定します。

SPRITE$（1）＝B$

とすると、スプライトパターン番号1に、B$に代入したスプライト面に表示するキャラクタが記憶されることになります。

SPRITE$（0）＝C$

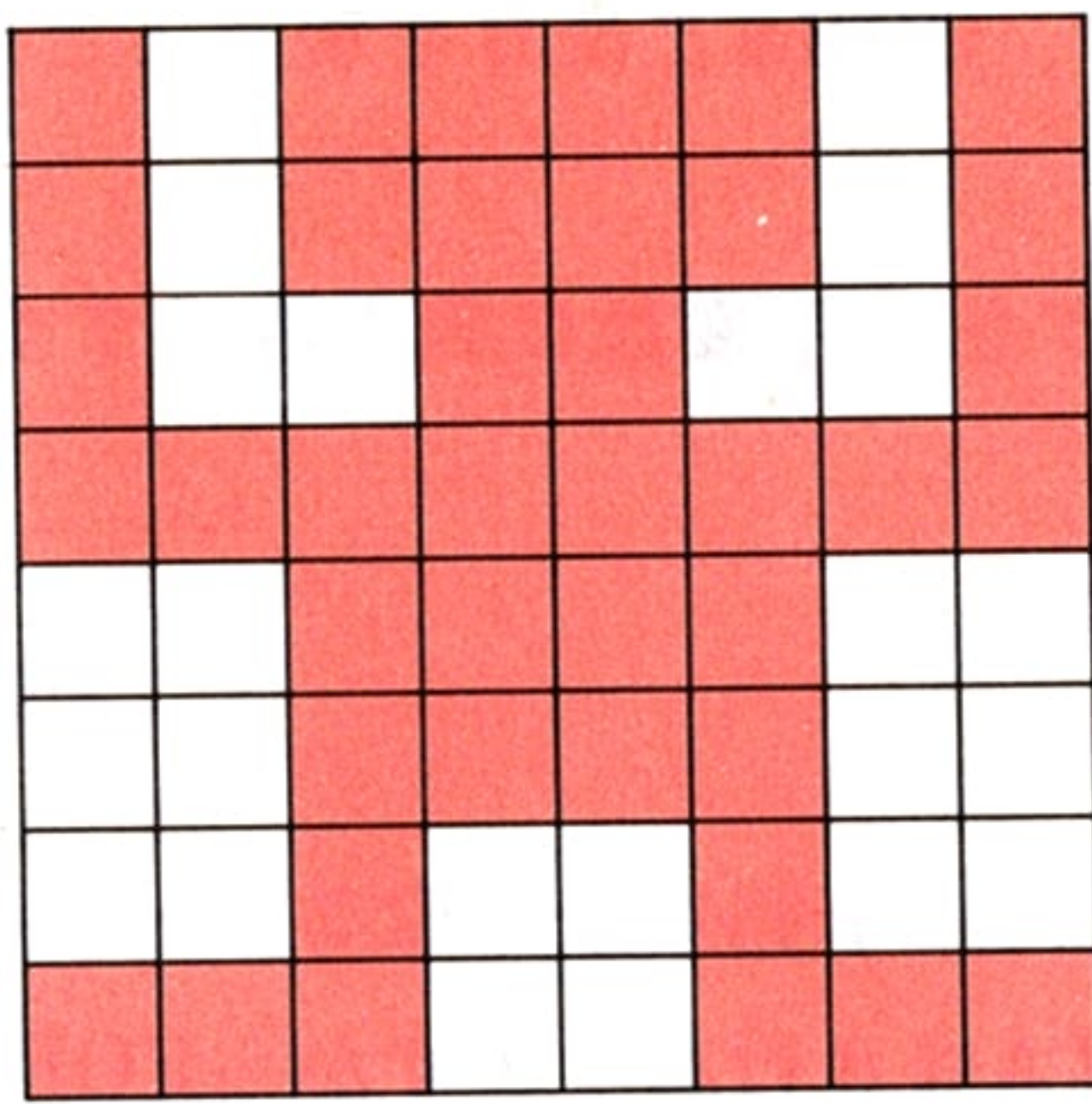
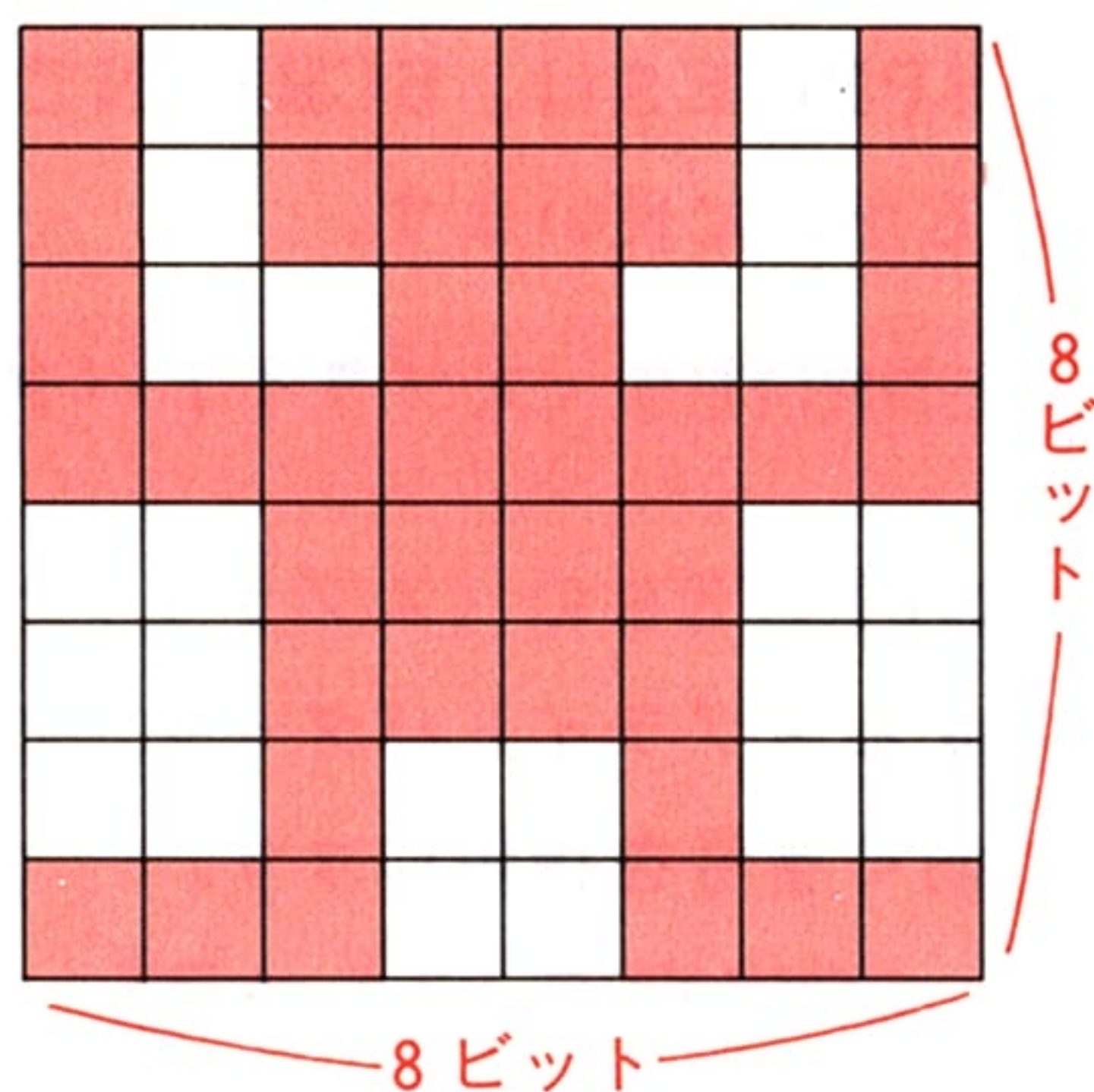
とすると、スプライトパターン番号0にC$に代入したスプライトパターンが記憶されることになります。

SPRITE$の〈スプライトパターン番号〉に与えることができる値は、SCREENの〈スプライトサイズ〉のところを0、あるいは1と指

定したときには、0から255までを指定することができます。ということは、最高256種類のキャラクタを記憶することができるということです。

SCREENの〈スプライトサイズ〉のところを2、あるいは3と指定したときは、SPRITE＄の〈スプライトパターン番号〉のところに与えることができる値は、0から63までとなります。これは、〈スプライトサイズ〉を2、3と指定したときは、0、1と指定したときよりも、大きいサイズのスプライトパターンを記憶するようになっているからです。

では、まずSCREENの〈スプライトサイズ〉を、0あるいは1に指定した場合から説明していくことにします。

▶SCREENの〈スプライトサイズ〉を0か1にしたとき

SCREENの〈スプライトサイズ〉を0か1に指定したときは、図に示すように横8ビット、縦8ビットの64ビットでスプライトパターンを作ります。〈スプライトサイズ〉を1に指定したときは、横8ビット、縦8ビットの64ビットで作ったパターンが、スプライト面に倍に拡大されて表示されることになります。

ここでは、図に示したような形をスプライトパターンとしてスプライト面に表示させるために、SPRITE＄で0から255までのスプライトパターン番号を指定して記憶させることにします。もちろんこのような図のままでは、SPRITE＄でスプライトパターン番号を指定して記憶させることはできません。まず、図を1と0の2進数で書き表わします。1と0の書き表わし方は、表示させる部分（赤い部分）を1で表わします。表示しない部分（白い部分）は0で表わします。する

と、つぎのようになります。

このように、図のなかの表示する部分と表示しない部分を1と0に変換したら、各行の1と0をカッコで囲んで、それにCHR＄と&Bをつけます。つぎに例として、1行目だけを示します。

CHR$（&B10111101）

CHR＄は、カッコのなかの1と0をキャラクタに変換します。&Bは、カッコのなかの数値が2進数であることを示します。

さて、図の表示する部分と表示しない部分を、CHR＄（&B 1 0 1 1 1 1 0 1）というように書き表わしたら、つぎは、これをデータとして、DATAに置きます。DATAに置くときは、CHR＄と&Bを取って置きます。

```
90 DATA 10111101
100 DATA 10111101
110 DATA 10011001
120 DATA 11111111
130 DATA 00111100
140 DATA 00111100
150 DATA 00100100
160 DATA 11100111
```

DATAには、CHR＄と&Bは取って置きましたが、このCHR$と&Bが必要なくなったわけではなく、このようにしたときは、READでデータを文字変数に読み込んだあとで、CHR$と&Bを使うことになります。

ではREADで、DATAに置いたデータを文字変数に読み取ります。ここでは、文字変数はA＄とします。

```
30 READ A$
```

READだけでは、1回しか働きません。1回に読み取るデータは1行目のDATAのデータだけです。DATAは8行ありますからFOR～NEXTでREADを8回繰り返えして、データを読み取ることになります。

```
20 FOR I=1 TO 8
30 READ A$
50 NEXT I
```

このFOR～NEXTの間には、行番号40としてもうひとつステートメントが入ります。つまり、READでA＄に読み取ったデータに&Bをつけて、データが2進数であることを示します。それを、VAL関数で数値に変換します。そして、VAL関数で変換したデータをCHR＄で、それぞれの行のデータをキャラクタにします。それから、キャラクタに変換した8個のデータをプラスして、別の文字変数に代入します。このようにしてスプライトパターンを作るのです。そのステートメントが、つぎに示すものですが、このステートメントはややこしいので、最初のうちはこのようにしてスプライトパターンを作る、というように、まるごとおぼえてしまったほうがよいでしょう。

```
40 B$=B$+CHR$(VAL("&b"+A$))
```

さてこのようにして、スプライトパターンをＢ＄に代入しました。つぎはSPRITE＄で、Ｂ＄に代入したスプライトパターンを、スプライト番号の何番に記憶させるか決めます。ここでは、スプライトパターン番号０番に記憶させることにします。Ｂ＄に代入したスプライトパターンを、スプライトパターン番号０に記憶させるには、SPRITE＄のカッコのなかを０として、＝のあとにＢ＄を置きます。

```
60 SPRITE$(0)=B$
```

これで、スプライト面に表示するスプライトパターンを、スプライトパターン番号０に記憶させることができたわけです。スプライトパターンをスプライトパターン番号１に記憶させるときは、SPRITE＄のカッコのなかを１、スプライトパターン番号10に記憶させるときは、SPRITE＄のカッコのなかを10にすればよいのです。

▶PUT SPRITEでパターンをスプライト面に表示

つぎは、このようにスプライト番号０に記憶させたＢ＄のスプライトパターンをスプライト面に表示させます。

それにはPUT　SPRITEを使います。

PUT SPRITE<スプライト面番号>,（X, Y）,
<カラーコード>, <スプライト番号>

PUT　SPRITEの〈スプライト面番号〉のところには、０から31までの値を指定します。これは、スプライトパターンを表示するスプライト面が、０から31までの32画面あるからです。１枚のスプライト面には、ひとつのスプライトパターンしか表示することができません。

（Ｘ，Ｙ）には、スプライトパターンを表示する座標位置を指定します。Ｘにはグラフィック座標の列番号、Ｙには行番号の指定です。〈カラーコード〉には、表示するスプライトパターンの色を指定します。

〈スプライトパターン番号〉には、スプライトパターンを記憶させた

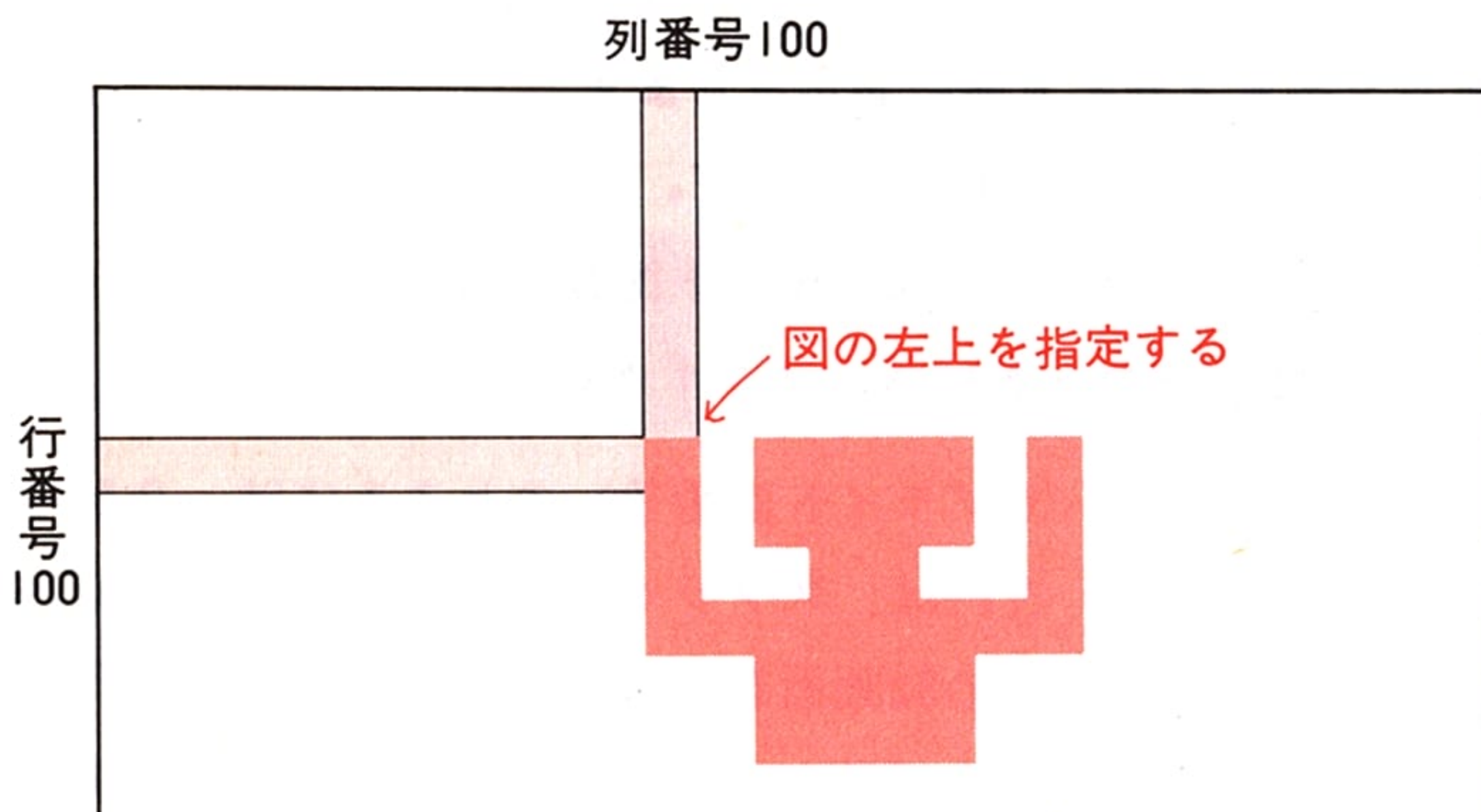
スプライトパターン番号を指定します。

ではまえにSPRITE＄で、スプライトパターン番号0に記憶させたスプライトパターンを、スプライト面番号4に表示させてみることにします。表示させる座標位置は、列番号100、行番号100の位置とします。表示させる色は赤にします。すると、PUT　SPRITEは、つぎのようになります。

```
70 PUT SPRITE 4,(100,100),8,0
```

これで、SPRITE＄でスプライトパターン番号0に記憶させたスプライトパターンが、スプライト面番号4に赤で表示されます。

ここでおぼえておくことは、スプライトパターンの表示の仕方です。

このようにすると、つぎの図に示すように、列番号100、行番号100の位置に、スプライトパターンの左上の1ビットが表示され、その位置から全体が表示されることになります。

プログラムを実行させるときは、つぎのように、行番号10として、SCREENで、画面モードとスプライトサイズを指定します。

```
10 SCREEN 1,1
```

そして、これまで説明してきた行番号20から160までをつづけて入力して実行させてください。なお、あいている行番号80には、ウエイト・ループを入れるのを忘れないでください(210頁参照)。

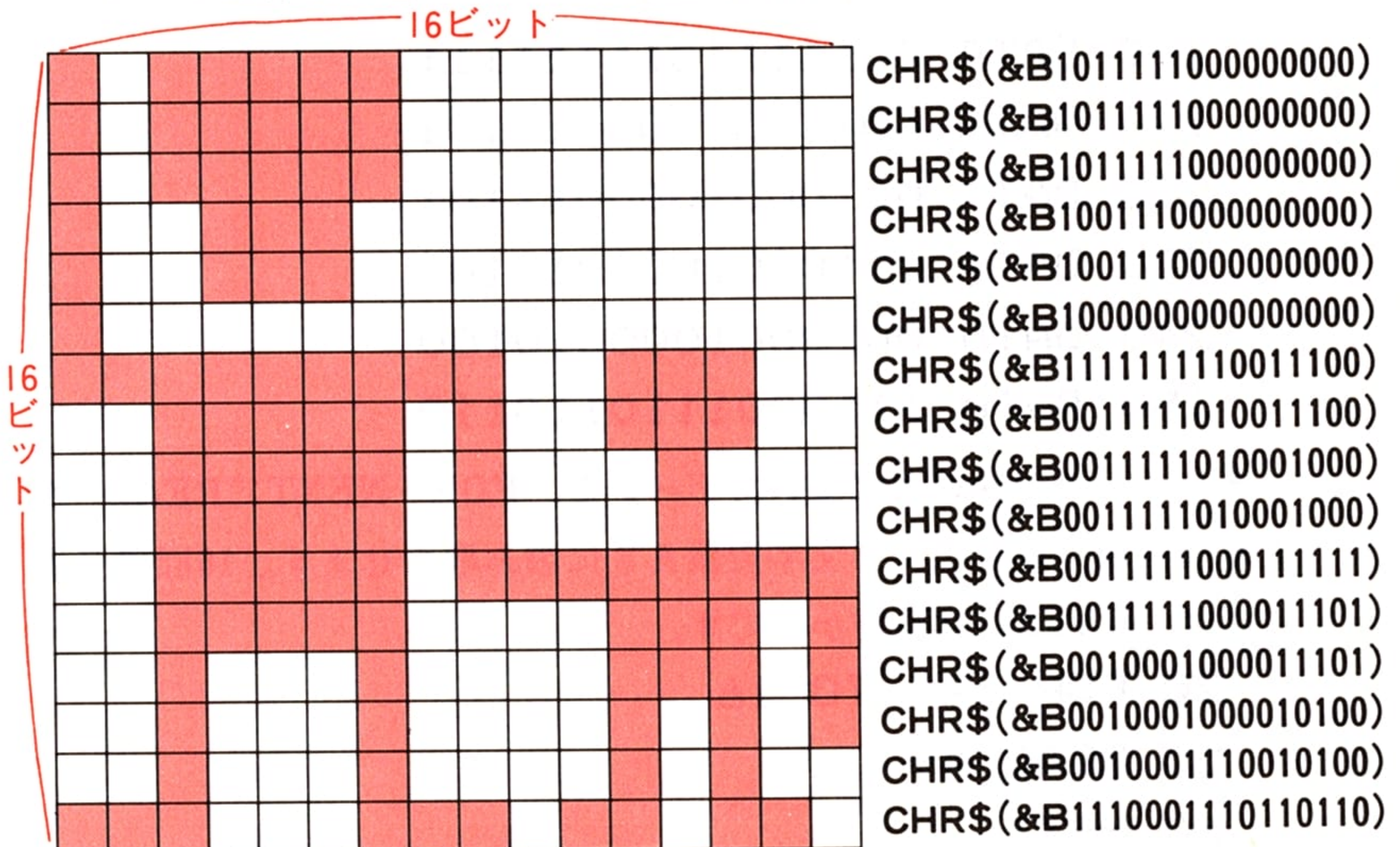
▶スプライトパターンの消去

さて、スプライト面にスプライトパターンを表示したあとで、スプライトパターンを消去することがあります。このようなときには、スプライト面にスプライトパターンを表示する座標位置の、行番号を示す値を209にします。

```
PUT SPRITE 4, (100,209)
```

このようにすると、スプライト面番号4に表示されたスプライトパターンが消去されます。

▶SCREENの<スプライトサイズ>を2か3にする

SCREENの<スプライトサイズ>を2か3に指定したときは、下の図に示すように、横16ビット、縦16ビットの256ビットで、スプライトパターンを作ることができるようになります。

SPRITE＄とPUT　SPRITE

まえに示した図のなかの表示する部分を1、表示しない部分を0にして表わすと、図の右に示したようになります。

この場合も、横8ビット、縦8ビットの64ビットの場合とおなじように、DATAにデータとして置きますから、CHR＄と&Bをつけないで、つぎのようにして置くことになります。

```
100 DATA 1011111000000000
110 DATA 1011111000000000
120 DATA 1011111000000000
130 DATA 1001110000000000
140 DATA 1001110000000000
150 DATA 1000000000000000
160 DATA 1111111110011100
170 DATA 0011111010011100
180 DATA 0011111010001000
190 DATA 0011111010001000
200 DATA 0011111011111111
210 DATA 0011111000011101
220 DATA 0010001000011101
230 DATA 0010001000010100
240 DATA 0010001000010100
250 DATA 1110001110110110
```

そして、このDATAに置いたデータを、FOR～NEXTでREAD　A＄を16回繰り返えして、文字変数A＄に読み取らせます。16回繰り返すのは、DATAが16行だからです。

```
20 FOR I=1 TO 16
30 READ A$
60 NEXT
```

FOR～NEXTの間には、まだステートメントが入ります。つまり、横８ビットのときのように、READでＡ＄にデータを読み取らせたら、そのデータに＆Ｂをつけて２進数であることを示し、それをVAL関数で数値に変換します。そして、CHR$でキャラクタに変換します。ただし、横16ビットの場合は、横８ビットの場合と違って、すごくややこしくなります。

横16ビットの場合、READで文字変数Ａ＄にデータを読み取らせたら、Ａ＄に読み取らせたそのデータを、８ビットずつのふたつに分けなければなりません。

1011111100000000

８ビット　　８ビット

このように左側８ビット、右側８ビットに分けて、別々にキャラクタに変換します。そして、８ビットずつ変換したキャラクタをプラス（連結）するのです。

ではまず、どのようにして文字変数Ａ＄に読み取らせた16ビットのデータを左側８ビット、右側８ビットのふたつに分けるかです。

Ａ＄に読み取られた16ビットのデータをふたつに分けるには、LEFT＄とRIGHT＄を使います。LEFT＄は、左から指定されただけの文字を取り出します。RIGHT＄は、逆に右から指定されただけの文字を取り出します。この場合は、Ａ＄に読み取られた16ビットのデータから、８ビットずつ取り出しますから、つぎのようになります。

LEFT$（A$, 8）

RIGHT$（A$, 8）

したがって16ビットのデータを８ビットずつのキャラクタに変換するステートメントは、つぎのようになります。

```
40 B$=B$+CHR$(VAL("&B"+LEFT$(A$,8)))
50 C$=C$+CHR$(VAL("&B"+RIGHT$(A$,8)))
```

このステートメントが、FOR～NEXTの間に入ります。READ A＄のつぎです。8ビットずつに分けてキャラクタに変換して、B＄とC＄に代入したら、つぎにプラス（連結）します。それと同時に、連結したキャラクタを、SPRITE＄で、指定したスプライトパターン番号に記憶させます。

```
70 SPRITE$(0)=B$+C$
```

ここでは、スプライトパターン番号0に記憶させています。

これが横16ビット、縦16ビットの256ビットのスプライトパターンを、SPRITE＄でスプライトパターン番号を指定して記憶させる方法です。このようにして、SPRITE＄でスプライトパターン番号0に記憶させたら、つぎはPUT　SPRITEで、スプライトパターンを取り出してスプライト面に表示します。この場合は、SPRITE＄でスプライトパターン番号0に記憶させたスプライトパターンを黄色で表示することにします。表示するスプライト面番号は6、表示させる座標位置は、列番号80、行番号30ということにします。

```
80 PUT SPRITE 6,(80,30),10,0
```

このプログラムを実行させるときは、行番号10として、SCREENで〈画面モード〉、〈スプライトサイズ〉を指定します。スプライトサイズは、スプライトパターンが横16ビット、縦16ビットですから2か3を指定します。たとえば、つぎのようにです。

```
10 SCREEN 1,3
```

そのあとに、行番号20から80までのステートメントをつづけます。行番号90としてウエイト・ループ、また行番号95として、210頁で説明したスプライトパターンの消去のステートメントを入れます。行番号95のあとに、行番号100から250までのDATAをつづけます。

MSX　ベーシック用語辞典

1986年5月25日　発　行　©

執筆代表　田　中　一　郎
　　　　　小　山　郁　夫
発 行 者　富　永　弘　一
印 刷 所　慶昌堂印刷株式会社

発行所　東京都台東区 台東2丁目24　株式会社 新星出版社
電話（831）0743　郵便番号110　振替東京 4-72233

ISBN4-405-06060-6

マイコン
シリーズ

まんが
パソコンゼミナール
きぎようへい・さとう光著

まんが
パソコンの一般知識
田中一郎・愛沢ひろし著

早わかり
ベーシック決まり文句
大橋均・田中一郎著

早わかり
マイコン用語辞典
関口泰・山科敦之著